Dhtml i JavaScript

Pàgines vives

Interactivitat

Formularis

Animació de capes

Control del temps

Persistència de dades

Reconèixer l'entorn

Autors: Artur Guillamet - Ramon Molins

Descàrrega

Octubre de 2006

Nota: Aquest document és una còpia en format Pdf del curs: "DHTML i JavaScript", creat l'any 2002 en format Web. L'original es troba disponible a:

http://www.xtec.cat/formaciotic/ateneu_historic/materials/td108/nucli/index.htm

Cerca Tots

m1	**Programar la web**
m1p1	Dreamweaver eficaç
m1p2	Organització del lloc
m1p3	Fulls d'estils amb estil
m1p4	JavaScript bàsic
m1ex	Estructura i portada del curs
m2	**Estructures JavaScript**
m2p1	Bucles for
m2p2	Recórrer matrius i objectes
m2p3	Bucles while
m2p4	Condicionals i selectors
m2ex	Menús d'enllaços
m3	**L'usuari en acció**
m3p1	Events i funcions
m3p2	Entrada de dades
m3p3	Variables i operadors
m3p4	Obre la finestra
m3ex	Comparacions aleatòries
m4	**Atenció a la diversitat**
m4p1	El model d'objectes (DOM)
m4p2	Detectar el navegador
m4p3	Afinem el estils
m4p4	Control de l'entorn
m4ex	Informe del client
m5	**El temps al meu costat**
m5p1	Rellotges
m5p2	Calendaris
m5p3	Barra de progrés
m5p4	Accions en segon pla
m5ex	Temps controlat
m6	**L'aspecte visual**
m6p1	Imatges i scripts
m6p2	Estils per a capes
m6p3	Animació
m6p4	Menús i persianes
m6ex	Efectes animats
m7	**Interacció i comunicació**
m7p1	Events i comportaments
m7p2	Validació
m7p3	La pàgina respon
m7p4	Interacció entre marcs
m7ex	Projecte interactiu
m8	**Calaix desastre**
m8p1	Persistència de dades
m8p2	Passar parámetres
m8p3	Fem aplicacions
m8p4	Reutilitzar el codi
m8ex	Pàgines amb memòria

Programar la web

En aquest curs estudiarem tècniques que ens han d'ajudar a millorar la funcionalitat de les pàgines web i dotar-les d'interactivitat. Haurem, doncs, de programar.

En principi, això pot espantar qui no ho hagi fet mai, per aquest motiu proposem un aprenentatge suau i assequible. Els conceptes, mètodes i estratègies que planteja el curs són una bona base per començar. I, si ja hem programat abans... tampoc no ens avorrirem.

Comencem aclarint conceptes.

Tu programes?

Un **programa** és una sèrie d'instruccions. Segur que algun cop hem programat el vídeo, la rentadora, o el microones. També programem aspectes relacionats amb la nostra feina, activitats, economia, ... d'alguna manera, tothom programa. Quan donem instruccions a un ordinador perquè executi una tasca, podem parlar d'un programa informàtic.

Per comunicar-nos amb l'ordinador hem d'utilitzar un **llenguatge** que la màquina pugui entendre. Si guardem les instruccions en un **arxiu**, podrem repetir el programa més vegades sense haver-les de tornar a escriure. Alguns programes que utilitzem habitualment tenen molts arxius, organitzats en contenidors (**carpetes**) que s'estructuren en **arbres**.

Els programes dissenyats per executar-se en un únic ordinador s'escriuen i es compilen, és a dir, s'empaqueten, juntament amb totes les instruccions i llibreries per crear un *executable* autònom. Tot el que necessiten per funcionar és al disc dur, de manera que van molt ràpids.

Xarxes, servidors i clients

Una xarxa és un grup d'ordinadors connectats que comparteixen informació i/o recursos. En xarxes locals ben organitzades la **velocitat** de comunicació és acceptable. En xarxes més grans i disperses intervenen molts factors que redueixen aquesta velocitat.

La solució adoptada, anomenada **Client-Servidor**, es basa en minimitzar el tràfic per la xarxa. El servidor (un programa, no una màquina) i el client (el navegador, un altre programa), es reparteixen la feina. Tots dos han d'estar d'acord en el llenguatge que utilitzen per donar-se instruccions mútuament i, alhora, ser capaços d'executar les que li envia l'altre.

Internet és, de moment, una xarxa lenta. El servidor pot ser ràpid, però pot rebre moltes peticions alhora i això el pot fer anar lent. En tot cas, el client sempre haurà d'esperar que les dades li arribin. La Xarxa és un tub estret i la solució és fer-hi passar poca cosa. A més, cada cop hi ha més serveis i usuaris.

Llenguatges de la web

L'objectiu inicial de la web era (encara ho és, amb matisos) posar informació a l'abast de la gent. Les unitats d'informació, les pàgines, només pretenien ser com les d'un diari o un llibre: en contingut i aspecte. Amb el temps i les possibilitats dels multimèdia han pogut evolucionar condicionades, però, a la velocitat de la tecnologia que les transmet.

HTML és el llenguatge de la web. Descriu els elements que formen una pàgina i permet enllaçar amb d'altres documents. Es basa en marques, que anomenem etiquetes (*tags*, en anglès). Si hem vist el codi font d'una pàgina, sabem que comença amb l'etiqueta <html> i acaba amb </html>. No és un llenguatge de programació. La seva sintaxi és molt senzilla, el vocabulari limitadíssim i tots els navegadors l'interpreten correctament.

CSS (fulls d'estils) és un complement de HTML que permet tenir més control sobre

l'aspecte i posició dels elements que formen la pàgina. Tampoc és un llenguatge de programació. Aporta els *identificadors*, que permeten tractar els contenidors de la pàgina com a objectes individuals, i les *classes,* que faciliten l'agrupament d'aquests objectes i la manipulació de les seves propietats. Pel que fa a compatibilitat, hi ha un mínim garantit i moltes diferències a partir d'aquest límit.

JavaScript és un llenguatge que s'utilitza per ampliar les capacitats de HTML i per inserir petites accions en pàgines web. Permet crear efectes especials i aporta interactivitat. No és un llenguatge de programació com ho són Java o C++, més aviat sembla un llenguatge de macros, com els de Flash o l'Office. El navegador és l'encarregat d'interpretar i executar les instruccions, que viatgen per la xarxa tal com s'han escrit.. Va néixer i ha evolucionat amb Internet, de manera que les diferències d'interpretació dels diversos navegadors i versions són notables.

Hi ha d'altres tecnologies que el client pot fer funcionar. La majoria es basen en petits programes (plug-ins) que s'han de tenir instal·lats. És el cas de Flash o dels *applets* de Java. Uns altres llenguatges poden executar feines al servidor: Perl, PHP, ASP, ... i un JavaScript per a servidors. Aquests dos grups, però, no formen part dels objectius del curs.

HTML Dinàmic

La utilització combinada de fulls d'estils, html i Javascript s'anomena HTML dinàmic, o DHTML, tot i que hi ha diversitat d'opinions sobre l'abast de la definició. Tots tres s'executen a la màquina client, de manera que el seu àmbit d'actuació queda ben clar ... i limitat.

En aquesta banda de la xarxa (la del client) hi ha, però, molta feina a fer. Es poden preparar les pàgines perquè "exprimeixin" els recursos que té el navegador: millorar l'aspecte, afegir funcionalitat, automatitzar processos, controlar el flux, prendre decisions. Quan s'utilitzen com cal aquests recursos, les pàgines semblen... més vives.

Per treballar amb DHTML cal conèixer bé els llenguatges i les possibilitats que ofereixen. Els editors de pàgines, com Dreamweaver o Frontpage, poden ajudar molt, però bona part de la feina s'ha de fer "a mà", escrivint el codi amb un editor de text.

Haurem de preparar un entorn ben estructurat i, si volem aprofundir i treure-li rendiment, també haurem d'estructurar i ordenar les nostres idees i mètodes de treball. L'estructura i l'organització han de ser, doncs, la base i, per aquí començarem.

A punt?

En aquest mòdul:

Pràctica 1: **Dreamweaver eficaç**
Pràctica 2: **Organització del lloc**
Pràctica 3: **Fulls d'estils amb estil**
Pràctica 4: **JavaScript bàsic**

Exercicis: **Estructura i portada del curs**

Fem aquest curs perquè sabem fer pàgines web i hem utilitzat el Dreamweaver. Si no és així, abans de començar hauríem de fer una bona repassada dels cursos **D72** i **D98** sobre Disseny i Creació de Pàgines Web. Sigui com sigui, donem per certa l'afirmació inicial.

Dreamweaver permet dissenyar pàgines utilitzant un editor visual. Per sota, genera codi HTML, css i JavaScript. Entendre aquest codi és fonamental si es vol anar més enllà.

Es pot aprendre molt si s'observa com treballa Dreamweaver, tot i que cal tenir present que només és un programa i que s'ha d'assegurar la possibilitat de tornar enrera (desfer). Sovint, genera més codi del que hauríem fet a mà, sobretot, quan crea scripts.

En aquesta pràctica, però, només treballarem sobre HTML. No es tracta de programar (tot arribarà), sinó d'utilitzar Dreamweaver de la forma més eficaç i, de passada, aconseguir que generi menys codi, i més net.

Farem l'esquema d'una pàgina utilitzant tècniques que ens permetin avançar amb rapidesa. Més endavant, aquest disseny ens servirà de base per aplicar nous conceptes.

Les etiquetes HTML més utilitzades

<head> </head> (cap), **<title> </title>** (títol), **<body> </body>** (cos), zones de la pàgina.
<table> </table> Defineixen l'inici i fi d'una taula.
<tr> </tr> cada fila d'una taula i **<td> </td>** cada cel·la individual dins d'una fila.
<p> </p> Defineixen un paràgraf.
** ** (negreta), **<i> </i>** (inclinada), **<u> </u>** (subratllada), etiquetes de format del text.
**
** (no té parella) Fa un salt de línia.
** ** Tipus de lletra, mida i color del text.
<img src="..."> Posa una imatge.

Algunes d'aquestes etiquetes tenen modificadors:
width i heigh estableixen l'amplada i l'altura.
color i bgColor defineixen el color del text i del fons.
align i valign permeten establir l'alineació horitzontal i vertical.

1.- Començar amb bon peu

Si encara no ho hem fet, hauríem de crear al nostre disc dur una carpeta on guardarem els treballs del curs. Més endavant farem tota l'estructura de treball de forma ordenada i ràpida. De moment, amb això n'hi ha prou. Farem tot el curs dins d'aquesta carpeta.

Prepararem una pàgina amb títol, sense marges i amb una taula..

Obrim el Dreamweaver i fem clic al botó "Mostrar vistas de código y diseño". Si el nostre monitor ens fa incòmoda aquesta vista, podem alternar entre les dues de les puntes, "código" i "diseño".

Observem que el programa ja ha generat l'estructura de la pàgina. Si la vista de codi no té activada la numeració de línies, anem al menú *Ver - Opciones de vista de Código - Números de líneas* i l'activem.

Ara anem al menú *Modificar - Propiedades de la página* i escrivim un títol per al curs, per exemple "Curs d108" o el que vulguem. A continuació posarem tots els marges a 0 d'una passada. Ens situem al quadre *Margen izquierdo*, i escrivim:

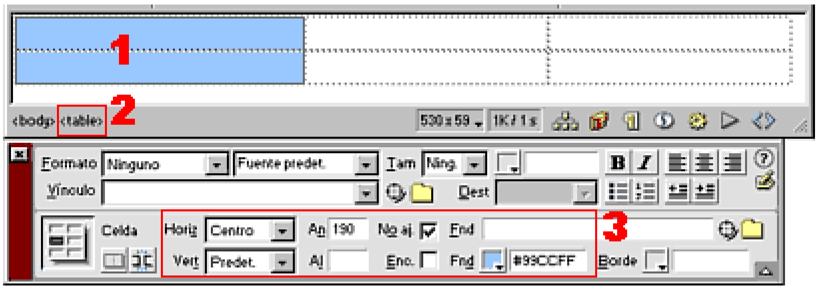
0 (tabulador) **0** (tabulador) **0** (tabulador) **0** (intro)

Inserim una taula de 2 files, 3 columnes, 770 píxels (resolució del monitor menys 30), relleno celda: 2, espacio celda: 0 i borde: 0. Observem el codi que ha creat.

2.- La barra d'estat, la finestra de propietats i el teclat

HTML treballa sobre contenidors. Si marquem una zona amb el mouse, Dreamweaver crearà un contenidor per poder-li assignar les propietats que especifiquem, normalment, un <div>. En molts casos això no és necessari, perquè els elements ja ocupen tot un contenidor.

Donarem format a la taula i en farem una còpia. Tindrem dues taules de la mateixa mida, l'una sota l'altra, de manera que l'aspecte final serà d'una única taula. Ho fem així per millorar el temps psicològic. Quan es carrega la pàgina des d'Internet, HTML no mostra les taules fins que troba l'etiqueta </table>. Seguint aquesta tècnica, la taula superior es veurà abans.



1.- Amb el ratolí, seleccionem les dues cel·les de l'esquerra.

2.- Comprovem que la taula **<table>** queda seleccionada a la barra d'estat i la línia fosca ocupa les dues cel·les. Dreamweaver aplica format als elements seleccionats.

3.- A la finestra de propietats, establim l'alineació, la mida i el color de fons. Com que volem que es mantingui la mida (190), seleccionem "No ajustar". Ignorem els botons d'alineació a la part superior dreta de la finestra de propietats. Els utilitzarem poc o gens.

Després fem clic sobre la paraula **<table>** a la barra d'estat. Premem *Control+C* (Copiar), *Fin* (desmarcar i situar-se darrera) i *Control+V* (Pegar). Tenim dues taules iguals.

Ens situem a la segona fila de la primera taula, fem clic **<tr>** a la barra d'estat i premem la tecla *Suprimir*. La primera taula ha quedat amb una única fila. Seleccionem les columnes segona i tercera, i les combinem. Ara té una fila amb dues cel·les.

3.- Taules enniuades

Observem un moment aquest disseny, perquè farem una pàgina similar. Donem un cop d'ull, també, al codi HTML que s'ha generat fins ara. De moment, l'hauríem d'entendre.

La primera taula serà el cap de pàgina. La fila inferior de la segona taula en serà el peu. La fila del mig (primera fila, segona taula) farà de contenidor dels menús, la zona de contingut i uns hipotètics enllaços (d'esquerra a dreta). Crearem, doncs, tres taules enniuades.

Seleccionem la cel·la de l'esquerra i la del centre (taula inferior, primera fila). A la finestra de propietats posem alineació horitzontal centrada i vertical superior. Després, ens situem a la cel·la de la dreta i li assignem alineació horitzontal a la dreta, vertical superior, 190 píxels d'amplada i marquem "No ajustar".

Tornem a la cel·la dels menús (esquerra). Inserim una taula de 8 files, una columna, 186 píxels, relleno celda: 1, espacio celda: 1 i borde: 0. Marquem l'última **<table>** a la barra d'estat i premem *Control+C* per copiar-la. Després, ens situem a la cel·la que contindrà els links (dreta) i premem Control+V per enganxar la taula que hem copiat.

A la cel·la de contingut crearem, de moment, una taula de 2 files, una columna i el 98%

d'amplada, sense borde. En aquesta cel·la no hem marcat "No ajustar", només ho fem a les que tenen amplada exacta. Més endavant, si cal, podrem esborrar o afegir files.

Donem format a les tres taules. Treballem sempre amb la barra d'estat i la finestra de propietats. Seleccionem grups de cel·les i apliquem format.

Quan acabem, tanquem el Dreamweaver i, quan ens demani si volem desar els canvis, contestem que sí i guardem l'arxiu a la carpeta rel del nostre lloc, amb el nom "index.htm". Ja tenim la primera pàgina. A la pràctica següent farem l'estructura.

4.- Codi net

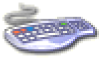
Ja ho hem dit, en aquest curs necessitarem entendre HTML. Dreamweaver, com a editor, fa exactament el que li manem. Això significa que si la nostra tècnica de treball es basava en seleccionar amb el ratolí i aplicar format, haurem de filar molt prim ... o canviar de tècnica.

Fins i tot quan seleccionem, per exemple, un bocí de text per canviar-li la font, mida o color, és convenient que ho fem d'una passada. Si no ens queda al nostre gust, el millor és desfer l'acció i tornar-ho a provar.

Quan hem de modificar una zona amb format establert abans, utilitzarem també la barra d'estat, seleccionant etiquetes com ara **<font>**, **<b>** o **<i>**. El programa marcarà tot el bloc i, així, no haurem de patir per si l'encertem o no.

Observar el codi amb regularitat és una bona prova per saber si treballem correctament. Quan veiem etiquetes duplicades (<font color ...> i, després, <font size ...>) o que no hem demanat (<div> i <span> són les més habituals), és que alguna cosa no va bé.

Generar codi net fa que les pàgines ocupin menys i es carreguin més ràpidament. A més, a més s'entenen millor i és més fàcil modificar-les o reutilitzar-les.

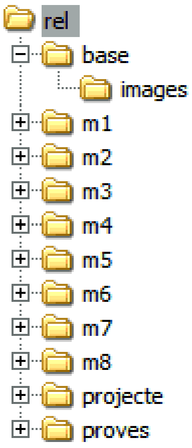


Un cop hem fet la pràctica, l'hauríem de repetir d'una passada fins que no ens calgui mirar l'explicació. Primer escrivim les mesures i alineacions en un full, o, millor encara, fem un esquema amb anotacions. Podem fer variacions en el disseny.

L'objectiu és saber fer l'esquelet d'una pàgina, net, en menys de cinc minuts.

Treballem més còmodes i millor quan ho tenim tot a l'abast i sabem on és. Com més complex és un projecte, més necessari és tenir un entorn ben estructurat. És com un edifici: primer els plànols, després l'estructura, els acabats, la decoració, ...

Aquest curs és un edifici moderadament gran, però d'estructura senzilla que ens ve marcada pels propis mòduls, pràctiques i exercicis. Perfecte! Per una banda, ens servirà d'experiència per a futures realitzacions i, per l'altra, ens facilitarà la feina i podrem dedicar el temps a allò que toca, en comptes de perdre'l buscant coses que hauríem de saber on són.



L'estructura de carpetes i subcarpetes serà semblant a la dels materials del curs, com és lògic. Si tenim aquests materials al disc dur o al CD, podem usar l'explorador per veure com s'han organitzat i com s'han anomenat els arxius. Nosaltres farem el mateix, però amb noms més curts, perquè no hi escriurem "d108".

L'ús de l'explorador de Windows (o similar) és imprescindible si volem entendre les estructures de carpetes i treballar-hi amb agilitat. Si no hi estem acostumats, hauríem d'aparcar temporalment aquesta pràctica i revisar el curs de Windows.

Elements HTML en aquest capítol

<script> i **</script>** delimiten un bocí de codi JavaScript.
** ** Delimiten un enllaç.

Deixem les instruccions JavaScript, perquè n'hi ha poques i senzilles, per més endavant.

1.- La pàgina per defecte

Com que volem crear una estructura buida, i que funcioni, haurem de tenir una pàgina per defecte, d'aquestes "en construcció" o similar. Podem fer-la tan sofisticada com vulguem. Aquí, però, en proposem una de ben senzilla, només l'adreça de l'arxiu.

Si encara no ho hem fet, hem de crear una drecera a la Llibreta del Windows dins de la carpeta "Windows \ SendTo", per tenir-la associada al menú del botó dret "Enviar a".

Obrim l'explorador, i ens situem a la carpeta de treball del curs i, a dins, en creem una de nova, anomenada "base". Ens hi situem. És buida. A la zona d'arxius (dreta de l'explorador) fem clic en el botó dret del ratolí i seleccionem "Nou - Document de text". L'anomenem "index.htm". Si el Windows ens avisa que hem canviat l'extensió, ho ignorem.

A continuació, sobre l'arxiu que acabem de crear, tornem a fer clic en el botó dret i seleccionem "Enviar a - Llibreta". Quan s'obri, escrivim el codi següent, després tanquem i guardem.

```
<script language="javascript">
document.write(location) ;
</script>
```

1.- Indiquem al navegador que comença un script.
2.- Escrivim l'adreça al document.
3.- Tanquem

2.- La carpeta base

És el moment de reflexionar. Què han de tenir en comú totes les carpetes? Perquè, per estalviar feina, si ja ho sabem, ho posarem tot a la carpeta base i, després, la clonarem per crear les altres.

En aquest curs no cal que ens hi trenquem massa el cap, és clar. En projectes més complexos, però, hi haurem de dedicar el temps i l'atenció que mereixi cada cas. Un bon plantejament d'aquesta carpeta ens pot estalviar moltes hores de feina.

Amb l'explorador, ens situem a la carpeta "base", al quadre d'arxius. Hi ha l'index.htm, que hem creat abans. Fem una nova carpeta per a imatges. Li podem posar "imatges" o "images" o "icons" o ... Si en tots els nostres projectes utilitzem els mateixos noms, ens resultaran més còmodes (la força dels hàbits és immensa). Aquí hem triat "images".

Parlant d'hàbits, com que construïm per a la web, ens hem d'acostumar a posar els noms d'arxius i carpetes en minúscules, i curts, fins a 8 caràcters. Aconseguirem la màxima compatibilitat amb tots els sistemes.

3.- Crear l'estructura

Un cop tenim una estructura base, podem copiar-la, enganxar-la, reanomenar-la, i en tindrem dues. Si ho anem repetint, en tindrem les que vulguem.

Aquesta tècnica de copiar i enganxar la utilitzarem molt. Convé, des d'ara mateix, que ens acostumem a fer-ho amb el teclat. És molt més ràpid:

Control + X = Tallar **Control + C** = Copiar **Control + V** = Enganxar

De vegades, l'estructura que en resulta pot ser, al seu torn, la base, o una de les bases, per fer-ne una altra de més complicada. Això, com veurem durant el curs, es pot aplicar en molts camps. Amb el temps, podem tenir biblioteques d'estructures, scripts, plantilles, recursos, ... que ens facilitaran la feina de dissenyar llocs web.

Situats a la rel del nostre lloc, marquem la carpeta base al quadre d'arxius (dreta) de l'explorador i la copiem. La desmarquem i, amb el botó dret del ratolí sobre una zona buida, l'enganxem 10 vegades.

Reanomenem les còpies "m1", ..., "m8", "projecte" i "proves". Estructura acabada. Cada carpeta, còpia de "base", té a dins una pàgina i la carpeta d'imatges. Comprovem-ho.

4.- Enllaçar les pàgines

Ara hem d'aconseguir que funcioni. La pàgina principal, encara buida, ha d'enllaçar amb totes les altres. L'editem amb el Dreamweaver i li afegim contingut. De moment, no ens fixem en el format. Això ho tractarem a la pràctica següent.

Editem la pàgina d'entrada, "index.htm" amb el Dreamweaver. Abans de modificar-la, definim el lloc a la finestra "Sitio".

A continuació ens situem a la primera cel·la de la taula destinada als menús, escrivim "Menú 1", marquem l'últim **<td>** a la barra d'estat i fem clic sobre la icona de carpeta de "Vinculo", a la finestra de propietats. Seleccionem l'arxiu "index.htm" de la carpeta "m1".

Amb el **<td>** encara marcat, premem *Control+C* (copiar), ens situem a la cel·la de sota, la marquem a la barra d'estat i premem *Control+V* (Pegar). Ara ens posem a la tercera cel·la, la marquem i tornem a prémer *Control+V.* Ho repetim en totes les cel·les.

```
<tr>
  <td bgcolor="#CCE4F8"><a href="m1/index.htm">Men&uacute; 1</a></td>
</tr>
<tr>
  <td bgcolor="#CCE4F8"><a href="m2/index.htm">Men&uacute; 2</a></td>
</tr>
<tr>
  <td bgcolor="#CCE4F8"><a href="m3/index.htm">Men&uacute; 3</a></td>
</tr>
```

Només ens queda canviar els títols i enllaços dels mòduls 2 a 8. Provem de fer-ho a la "Vista de código" del Dreamweaver. Ens familiaritzarem més amb el llenguatge i és més ràpid que no pas fer-ho amb l'editor.

5.- Prova pilot

Si tot ha sortit com cal, la nostra pàgina d'entrada enllaça amb tots els "index.htm" de les carpetes. Deixem l'enllaç amb el projecte per a l'exercici d'aquest mòdul.

Podem fer una prova per veure com la pàgina d'entrada carrega els mòduls. Com que els

hem fet amb Javascript, cada un mostra l'adreça de la pàgina. Naturalment, ho hem de provar amb diversos navegadors.

Si tenim la intenció de guardar l'estructura per poder-la reutilitzar, ara és un bon moment. Podem comprimir la carpeta amb el Winzip i guardar-la. Anomenem l'arxiu, per exemple, "curs8mod.zip" i així sabrem que es refereix a un curs de 8 mòduls.

També podem esperar, perquè, a la pràctica següent, li aplicarem estils.

HTML té greus limitacions a l'hora d'aplicar el format a una pàgina. La seva estructura seqüencial obliga a formatar elements individuals, i no es poden tractar en bloc. Es va dissenyar en entorns científics i acadèmics, on importava molt el contingut i poc la forma.

Els fulls d'estils CSS han solucionat aquest problema, i aporten quelcom més:
- Podem aplicar format a un lloc, una pàgina, elements concrets o una única etiqueta.
- Permeten tenir maquetacions diferents per a la pantalla i la impressora.
- Tenen moltes més possibilitats i apliquen el format d'una manera més fina, més exacta.

ab ab ab ab ab ab ab **ab ab ab** ab ab ab ab

Els navegadors entenen CSS a partir de la versió 4 (Internet Explorer, a partir de la 3, amb limitacions), però no tot ho entenen igual. Per això, s'han de manejar amb cura i reserves, o hi pot haver usuaris que no vegin el format ... o la pàgina.

En general, els navegadors actuals mostren; totes les propietats que definim als estils, si ho fem bé, respectarà l'herència al peu de la lletra. Netscape 4.7, en canvi, és tota una aventura, perquè no sabrem si alguna cosa funciona fins que no ho provem. Cal, doncs, fer moltes proves per tal d'assegurar que tothom veurà, més o menys bé, els nostres dissenys.

Elements CSS en aquest capítol

font-family : estableix l'ordre de prioritats en l'ús de fonts per a un element.
color : color del text. Pot usar valors hexadecimals (#FFFFFF) o literals (white).
text-decoration: decoració del text: res, subratllat, intermitent, ...
font-size: mida de la lletra. Es pot especificar en diverses unitats.
width: defineix l'amplada que ocuparà un element.
background-color: especifica el color de fons.

1.- Jerarquia

HTML i CSS són sistemes basats en una jerarquia. Això vol dir que uns elements en contenen uns altres que, al seu torn, en contenen d'altres. No és estrany, els sistemes jeràrquics són abundants, en aquest entorn, només cal que ens fixem en com s'estructura un lloc web, una xarxa o el nostre disc dur.

La jerarquia va de contenidor a contingut, de general a concret. L'element més alt de l'escala és el cos del document (body), i a partir d'aquí, van derivant els altres. Les característiques s'hereten. Això té molts avantatges, i algun inconvenient.

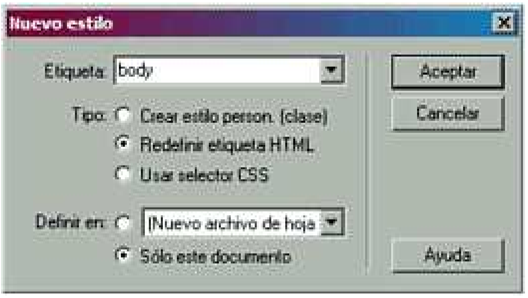
Triar una bona manera de definir els estils és important perquè pot condicionar les fases posteriors del disseny i, si prenem una decisió equivocada, podem hipotecar possibilitats que, d'una altra manera, tindríem a l'abast.

Si redefinim una etiqueta HTML, - per exemple "table", un contenidor en la part alta de la jerarquia -, ens veurem limitats per dues bandes: quan vulguem tenir una versió diferent d'aquest contenidor i quan intentem aplicar format als elements que conté.

La solució, normalment, passa per definir pocs estils i fer-ho de manera que als més generals els apliquem opcions generals. Per exemple podem definir la font al <body>, les mesures als estils personalitzats i aplicar colors directament amb HTML.

Amb la jerarquia comencen les incompatibilitats. Mentre Explorer i Mozilla la respecten fidelment, Netscape 4.7 no ho fa, o ho fa parcialment i en segons quines versions. No accepta, per exemple, la definició d'una font per al cos, de manera que l'haurem d'assignar a totes les classes, una per una.

Obrim la pàgina d'entrada ("index.htm" de la rel) amb el Dreamweaver i anem al menú "Texto - Estilos CSS - Editar hoja de estilos..." o, més ràpid, premem *Control+Majús+E*.



Marquem "Redefinir etiqueta HTML", "Sólo este documento" i seleccionem **body**. Li apliquem només el tipus de lletra i acceptem. A continuació redefinim l'etiqueta **a** (els enllaços). Aquí triem colors (lletra i/o fons) i decoració.

Passem als estils personalitzats, pocs. En aquest cas en farem quatre: **.titol**, **.menu**, **.normal** i **.lnk**. No utilitzem la paraula "link" perquè és reservada ni "enllaç" perquè té una "ç". En alguns sistemes podrien funcionar, en altres donarien problemes.

Què definim en aquests estils? Poc, bàsicament la mida de la lletra. També la font, si ha de ser diferent de la general definida per al cos. El nombre ideal de fonts és d'una o dues en tot el lloc web, tot i que, en alguns casos, se'n poden necessitar més.

Utilitzem les fonts que proporciona Dreamweaver, per exemple, Arial per a fonts grans i mitjanes i Verdana per a les petites. Establim les mesures en píxels: no més de 20 ni menys de 10, de manera que siguin llegibles sense arribar a molestar.

Evidentment, com s'ha dit abans, si volem que Netscape 4.7 mostri un estil personal amb una font determinada, li ho hem de dir, perquè no heretarà pas la que hem definit al body.

2.- Comportament dels enllaços

Hem redefinit els enllaços. Amb això n'hi hauria prou per a l'antic navegador de Netscape, però els navegadors actuals tenen més possibilitats, i les volem aprofitar. A més, potser ens interessa que algunes zones (per exemple, els menús) tinguin un aspecte o comportament diferents.

Dreamweaver no facilita la definició d'aspecte per als enllaços dels estils personalitzats, però tampoc ho prohibeix. Només cal saber com es fa.

El següent pas serà "Usar selector CSS". Triem **a:hover** i definim colors i/o decoració. Com que al pas anterior havíem redefinit els enllaços (a) en lloc d'usar el selector (a:link), ens hem estalviat dues definicions. Si ho volem, però, també podem definir els **a:active** i **a:visited**, tot i que no és imprescindible.

A l'hora de definir enllaços especials, tornem a "Usar selector CSS" i, en lloc de triar, escrivim **.menu a** per tal d'establir l'aspecte dels enllaços dels menús. Utilitzarem un petit truc que, com passa sovint, només funciona amb l'Explorer. Es tracta d'aconseguir que l'enllaç ocupi tota la cel·la. Per això, ens dirigim a la secció "Cuadro" i establim un "Ancho" del 100%.

Finalment, definim **.menu a:hover** i li assignem colors de lletra i fons que contrastin amb l'estat normal dels menús. Tindrem un aspecte similar als menús de Windows.

Un cop acabada la definició d'estils, els revisem des de la "Vista de código" per entendre com ho ha fet el Dreamweaver i quina codificació ha utilitzat. Normalment, ho haurà situat a la secció <head> del document:

```
<style type="text/css">
<!--
body { font-family: Arial, Helvetica, sans-serif}
a { color: #003366; text-decoration: none}
a:hover { color: #990000}
.titol { font-size: 18px; font-family: Arial, Helvetica, sans-serif}
.menu { background-color: #CCDDEE; font-size: 12px; font-family: Arial, sans-serif}
.normal { font-size: 10px; font-family: Arial, Helvetica, sans-serif}
.lnk { font-family: Verdana, Arial, sans-serif; font-size: 12px}
.menu a { width: 100%}
.menu a:hover { color: #FFFFFF; background-color: #990000}
-->
</style>
```

3.- Aplicar estils a contenidors

Els estils s'apliquen a les etiquetes, als contenidors, com fèiem al format HTML (pràctica 1), de manera que haurem d'actuar igual. Treballarem, doncs, amb la barra d'estat.

Es tracta d'aplicar estils amb el mínim de passos possibles, i aconseguir que Dreamweaver no ens faci cap contenidor nou no desitjat. Utilitzarem la finestra d'estils CSS.

Podem, com s'ha vist en pràctiques anteriors, seleccionar grups de cel·les amb el ratolí o marcar directament les etiquetes a la barra d'estat. En qualsevol cas, les cel·les són els únics contenidors on aplicarem estils en aquesta pràctica.

Com que ja tenim la base suficient per treballar així, l'explicació d'aquesta tasca es queda intencionadament curta.

Obrim la finestra d'estils. Marquem la zona on volem aplicar un estil i el seleccionem de la llista. Un cop hem establert els estils a tota la pàgina, tanquem la finestra i comprovem que funcionen amb tots dos navegadors.

4.- Estils segons el medi

L'etiqueta de definició d'estils permet altres modificadors que Dreamweaver no contempla. Un dels més interessants és aquell que ens permet establir un conjunt d'estils per a un medi específic. Es tracta del modificador **media**:

<style type="text/css" media="screen"">

Com a valors, podem posar-hi: **all** (tots, predeterminat), **screen** (pantalla), **print** (impressora), **tv** (endevina-ho), **projection** (projector), **handheld** (ordinadors de butxaca), **braille** (dispositiu tàctil), **aural** (sintetitzador de veu), **tty** (teletip) ... En properes versions, la llista es pot ampliar.

Sovint, el disseny d'una pàgina és per ser visualitzat en pantalla, i resulta horrorosa quan s'imprimeix o, senzillament, no es veu. Imaginem una pàgina amb lletra blanca i amb una imatge fosca de fons. Quan s'intenta imprimir amb un navegador que té la impressió de fons desactivada, no es veu res. Una bona opció és preparar dos fulls d'estils en arxius independents i enllaçar-los a la capçalera del document:

<link rel="stylesheet" href="pantalla.css" media="screen" type="text/css">
<link rel="stylesheet" href=".impre.css" media="print" type="text/css">

Amb aquesta tècnica podem assegurar la presentació en pantalla i la impressió, de manera que es vegi perfectament en tots dos medis. De passada, si enllacem els mateixos arxius en totes les pàgines del nostre lloc (en aquest curs ho hem fet així), tindrem el manteniment més fàcil perquè, quan vulguem fer modificacions d'aspecte, només haurem de revisar dos arxius.

En aquesta pràctica iniciarem els primers passos amb JavaScript. A diferència de les tres anteriors, que tenien com a objectiu preparar l'espai de treball que utilitzarem durant tot el curs, aquí començarem a veure els codis JavaScript més senzills.

Farem el primer script analitzat detalladament i veurem les diferents formes d'introduir codi JavaScript dins la pàgina HTML.

També estudiarem com podem interactuar amb el programa, mitjançant senzills exemples per introduir dades i obtenir els resultats.

Aquests són exemples bàsics que es tornaran a utilitzar al llarg de tot el curs formant part d'altres conceptes més avançats.

Elements JavaScript en aquest capítol.

Mètode **document.write() :** Introdueix un text en el document actiu.
Mètode **document.writeln() :** Introdueix un text i un salt de línia en el document actiu.
Mètode **prompt() :** Mostra un quadre de diàleg que fa una consulta a l'usuari.
Mètode **alert() :** Mostra un missatge d'alerta a l'usuari.
Declaració **var :** Declara una variable.

1.- El meu primer programa en JavaScript.

El codi JavaScript s'inclou entre les directives:

```
<script language="JavaScript">
   .....
</script>
```

El codi que es trobi dins aquestes directives s'executarà en el moment que la càrrega de la pàgina llegeixi el seu contingut.

Observeu el paràmetre language="JavaScript". Això indica al navegador que el codi correspon a JavaScript. Si fos convenient, també podríem indicar la versió implementada, per exemple: "... JavaScript1.2", "... JavaScript1.3", etc.

Hi ha d'altres llenguatges de programació que es poden indicar en aquest mateix paràmetre, per exemple "VBScript". Aquest és el llenguatge de programació implementat per Microsoft i derivat del Visual Basic. Solament és reconegut pel navegador Internet Explorer i en aquest curs no l'estudiarem.

Les directives <script> ... </script> s'han d'incloure dins la secció de capçalera <*head*> i/o la secció de cos <*body*> de la pàgina HTML.

A continuació, crearem una pàgina HTML que en el procés de càrrega executarà una instrucció en JavaScript que escriurà "Hola, món". Podeu utilitzar el Dreamweaver, amb la "Vista Código" o la llibreta per crear la pàgina d'exemple. El codi és el següent:

```
<html>
<head>
</head>
<body>
   <script language="JavaScript">
     <!--
       document.writeln("Hola, món");
     //-->
   </script>
</body>
</html>
```

Quan es carrega aquesta pàgina en el navegador s'executa el codi que es troba entre <script> i </script>, la seva funció és molt simple: escriure en el cos del document la frase "Hola, món". D'aquesta tasca s'encarrega el mètode **writeln()** de l'objecte **document** (En mòduls posteriors, aquest objecte l'estudiarem amb més detall).

Situat dins els parèntesi de **writeln()** es troba la frase que volem escriure a la pàgina web. Aquesta és una *cadena* de text i per tant es troba situada entre cometes ("). Observeu com les paraules *document* i *writeln()* es troben unides mitjançant un punt.

En l'argot informàtic s'entén per *cadena* un conjunt de caràcters alfanumèrics, habitualment inclosos entre cometes, és a dir, un text. Per tant, no és un valor que pugui ser avaluable numèricament. També s'anomena amb el seu nom equivalent *String*.

També es pot observar com el codi ha estat introduït entre els símbols <!-- i --> que corresponen als elements delimitadors dels comentaris en HTML, això es fa d'aquesta forma per evitar que els navegadors de versions antigues, o actualitzats amb JavaScript desactivat per l'usuari, no detectin aquest codi ja que no el podrien interpretar.

Ara bé, utilitzant els navegadors actualitzats no hi ha cap inconvenient en no utilitzar els símbols delimitadors de comentari, ja que els navegadors moderns interpreten correctament el codi JavaScript i rares vegades l'usuari desactiva aquesta opció. De fet, al llarg del curs, trobarem molts exemples que ja no contenen els delimitadors de comentari HTML al voltant del codi JavaScript.

Davant de **-->** hi ha situades dues barres (***//***). Aquestes dues barres corresponen als comentaris de JavaScript. Tot el que es trobi a continuació de les dues barres fins el final de la línia serà considerat com un comentari. En aquest cas, són utilitzades per evitar que JavaScript interpreti erròniament el finalitzador de comentari HTML (**-->**).

També podem indicar comentaris JavaScript de més d'una línia situant-los entre els símbols /* .... */. Aquest tipus de comentaris són els originals de C.

La línia document.writeln(....); està finalitzada amb un punt i coma **;** . Aquest element serveix per indicar el final d'una instrucció JavaScript, però, no és imprescindible si al final de la instrucció hi ha un salt de línia, com en l'exemple. En cas de tenir més d'una instrucció en la mateixa línia, és imprescindible el **;** per separar-les.

Al llarg del curs trobarem codis JavaScript que contenen el **;** i d'altres que no. No té massa importància, ja que és opcional si solament hi ha una instrucció per línia. Ara bé, trobarem molts altres codis que sempre el contenen, el motiu és per analogia amb altres llenguatges de programació com el C, Java, etc. en els quals és obligatori sempre la utilització del **;** .

2.- Interacció amb l'usuari.

Un dels principals objectius de JavaScript utilitzat dins una pàgina HTML és interaccionar amb l'usuari. Això implica rebre una informació, processar-la i posteriorment mostrar els resultats.

Hi ha moltes formes d'introduir i mostrar la informació que es veurà al llarg del curs, a continuació veurem les formes més elementals mitjançant quadres de diàleg.

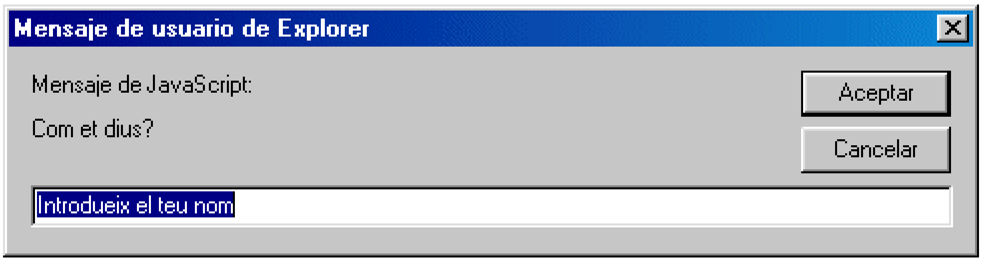
Introducció d'informació
per introduir una dada podem utilitzar el mètode: **prompt()**. Aquest mètode obre un quadre de diàleg que ens fa una consulta. La seva sintaxi és:

```
prompt("Missatge informatiu", "Resposta per defecte");
```

Si la invoquem segons el següent exemple:

```
resposta = prompt("Com et dius?", "Introdueix el teu nom");
```

ens mostrarà un quadre de diàleg com el següent:



Després de prémer sobre "*Aceptar*", el valor introduït al quadre de diàleg quedarà assignat a la variable **resposta** com una cadena de caràcters.

Una altra forma d'introduir informació és mitjançant el mètode **confirm()**. La seva utilització detallada la podeu veure al mòdul 2 pràctica 4.

Mostra de resultats

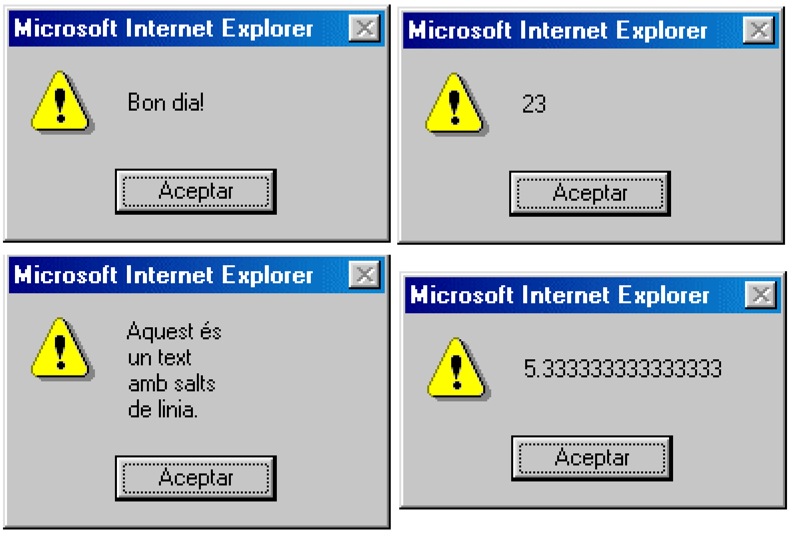
Si el que desitgem és treure informació per a l'usuari, es pot utilitzar el mètode **alert()**. La seva sintaxi és la següent:

```
alert(missatge);
```

Per exemple podem invocar-la de les següents formes:

```
alert("Bon dia!");
alert(23);
alert("Aquest és\nun text\namb salts\nde línia.");
alert(8*2/3);
```

es veuran unes finestres de diàleg com les següents:



Observeu com en la tercera forma, la que conté un text amb salts de línia, es provoca el salt de línia introduint **\n**, en els llocs apropiats entre mig de la frase. Si es vol mostrar una frase ho fem indicant-la entre cometes **" "**, en canvi, si es vol mostrar un valor numèric o el resultat numèric d'una operació aritmètica no s'ha de posar entre cometes **" "** .

També es pot treure informació directament a sobre de la pàgina HTML que conté el codi mitjançant els mètodes d'escriptura de l'objecte document.

Els més habituals són les següents (s'ha utilitzat *writeln()* a l'exemple de l'apartat 1):

document.write() --> Escriu text i codi HTML en el document.
document.writeln() --> Escriu text i codi HTML, més un salt de línia, en el document.

El següent exemple mostra com podem produir un diàleg interactiu amb l'usuari mitjançant quadres de diàleg.

```
<html>
<head>
</head>
<body>
<script language="JavaScript">
  <!--
    var resposta;
    resposta = prompt("Com et dius?", "Introdueix el teu nom");
    alert("Hola " + resposta);
  //-->
</script>
</body>
</html>
```

En aquest codi es declara inicialment una variable de nom **resposta**.

```
var resposta;
```

Seguidament s'invoca el mètode **prompt()** amb dos paràmetres, el primer és la pregunta que es mostrarà en el quadre de diàleg ("*Com et dius*") i el segon paràmetre ("*Introdueix el teu nom*") representa la resposta per defecte.

```
resposta = prompt("Com et dius?", "Introdueix el teu nom");
```

L'usuari té l'opció de fer clic sobre dos botons: *"Aceptar"* i *"Cancelar"*. El primer botó envia com a retorn d'aquest mètode el valor que es troba dins el quadre de text i el segon botó envia el valor *"null"*, tant en un cas com en l'altre, el valor retornat s'assigna a la variable **resposta**.

Finalment, la línia:

```
alert("Hola " + resposta);
```

mostra un missatge d'alerta que conté el text format per la cadena *"Hola "* concatenat (+) amb el contingut de la variable **resposta**.

3.- JavaScript dins un link.

Una altra forma d'invocar l'execució d'instruccions JavaScript és mitjançant un **link**. En aquest cas el codi pot estar inclòs dins el paràmetre **href**. En el següent exemple veiem com s'invoca l'execució del mètode **alert()** cada cop que es fa clic sobre l'enllaç.

Observeu com el link comença per : 'javascript : ....' . Aquesta paraula indica que a continuació no hi ha un hiperenllaç normal, sinó, un codi JavaScript que s'ha d'executar. Molts navegadors interpreten correctament el codi JavaScript sense aquesta paraula, per tant, en moltes ocasions ja no s'utilitza i es posa el codi directament.

Aquesta forma d'executar codi JavaScript també es pot utilitzar per invocar funcions d'usuari més complexes.

```
<html>
<head>
</head>
<body>
  <a href='javascript:alert("Hola, món");'>Clica'm a sobre</a>
</body>
</html>
```

Programar la web > Estructura i portada del curs

En aquest mòdul hem vist de quina manera podem organitzar un lloc, fer que funcioni a partir d'una pàgina d'entrada i aplicar-li format amb fulls d'estils. Tot això de forma ràpida, neta i organitzada. Com a exercici, farem la revisió i ampliació de la pàgina d'entrada, posarem els estils en un arxiu independent i ho enviarem a la nostra carpeta al servidor.

Exercici

El primer pas serà separar els estils de la pàgina principal, de manera que els puguem enllaçar des de tot el lloc i, si cal, canviar l'aparença de totes les pàgines de cop. Una de les maneres més ràpides per fer-ho és:

1. Amb l'Explorador, ens situem sobre la pàgina d'entrada "index.htm", premem el botó dret i triem "Enviar a - Llibreta". Marquem tot el bloc d'estils (entre <style ...> i </style>) i premem *Control+X* (Cortar). Tanquem l'arxiu i guardem els canvis.

2. Fem un nou arxiu i l'anomenem "pantalla.css". L'obrim i premem *Control+V* (Pegar). Eliminem les dues línies HTML del principi i del final. Deixem només les línies que contenen la definició d'estils. Tanquem i guardem.

3. Obrim "index.htm" amb el Dreamweaver, anem al menú "*Texto - Estilos CSS - Adjuntar hoja de estilos...*" i seleccionem la que acabem de crear.

La pàgina torna a tenir els estils, ara en un arxiu independent. Aprofitem que la tenim oberta per afegir-hi contingut, enllaços, capçalera, peu, el nostre correu, ... i un enllaç cap al projecte, que l'havíem oblidat. Mireu com ho ha fet en Bill.

A la zona de contingut podem posar el text i les imatges que vulguem, per exemple: sobre nosaltres i les nostres motivacions per fer aquest curs (original, oi?). A la zona d'enllaços, els dels materials del curs i les adreces relacionades, les nostres pàgines a internet, ...

Com a capçalera i peu, per exemple: el títol del curs i la nostra adreça de correu. És un bon costum afegir-hi també un enllaç a l'adreça que tenim a Internet. Així, si algú mira la pàgina descarregada en local, sempre podrà accedir a la versió actualitzada de la xarxa.

Un cop realitzada la feina hem de crear el nostre lloc a Internet (última pràctica del primer mòdul del curs d98), i transferir-hi l'estructura i el contingut del lloc local i, si volem, les carpetes "base" i "proves".

Recomanacions

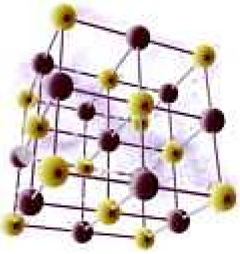
Hem de revisar el codi per tal que no hi restin instruccions de format innecessàries, perquè és això, precisament, el que es valorarà.

És evident que el model exposat en aquest mòdul és només un exemple. Podem fer el disseny que més ens agradi. El més important és seguir el mètode.

Envieu un missatge a la vostra tutoria un cop la web estigui penjada a Internet. Indiqueu sempre l'adreça de forma que s'hi pugui accedir directament.

Estructures JavaScript

A partir d'aquest mòdul, comencem a programar. Això vol dir que l'editor, Dreamweaver o qualsevol altre, no ens podrà ajudar. I, encara que ho pogués fer, no seria convenient, perquè intentem aprendre un llenguatge, i l'hem de "xerrar" (amb una màquina!!!). Tot i això, si utilitzem Dreamweaver tindrem una eina valuosa. En parlarem més endavant.



Tal com hem vist, JavaScript executa **instruccions** sobre unes **dades**. Si li diem "escriu" també li hem de dir què volem escriure, o si li diem "fes-ho **x** vegades", hem d'especificar el valor de **x** i l'acció que ha de fer cada vegada. Aquesta acció, al seu torn, haurà d'actuar sobre unes altres dades i, potser, executar altres accions. Tornem a ser davant d'una estructura.

Hi ha **dades** que tenen una utilitat immediata i s'escriuen associades a una instrucció, per exemple, *alert("Hola món")*. D'altres, però, s'han de guardar per tal de poder-les reutilitzar, o perquè serveixen per controlar el flux del programa. Aquestes s'assignen a **variables**.

L'autèntica potència d'un llenguatge resideix en les **instruccions** que permeten repetir operacions, o desviar el flux del programa, en funció dels valors guardats en variables. En aquest mòdul aprendrem què fan i com s'utilitzen.

Potser, si no hem programat mai, ens resulti difícil d'assimilar, en principi. Com engegar un cotxe per primera vegada costa amunt : molta força, poca velocitat. Després, però, posarem les altres marxes i tot serà diferent. Cal que coneguem els conceptes i elements del llenguatge d'aquest mòdul, ja que són bàsics i els tornarem a veure moltes vegades durant el curs.

Bucles

Un bucle és una feina que es repeteix mentre es compleixi una condició. El dia, l'any, les nostres rutines, el mateix sistema de numeració, són bucles. Qualsevol tasca que s'hagi d'executar repetidament és un bucle.

Les pàgines web també tenen estructures repetitives. Per exemple, la nostra pàgina principal té enllaços cap a tots el mòduls. Si diem "Fer els enllaços des del mòdul 1 fins al 8" ho entenem perfectament. Al disseny, però, hem escrit vuit vegades les instruccions per crear files, cel·les i enllaços. És evident que es pot millorar.

Condicionals

"Si et portes bé, tindràs un regalet. Si no, carbó.". Estem acostumats a les condicions. Es tracta de fer una comparació: si el resultat és cert, s'actua en un sentit i, si és fals, en un altre. "O blanc, o negre", diem, i hem de prendre una decisió.

Amb els condicionals atorguem a les pàgines la capacitat de decidir i, en conseqüència, d'adaptar-se a l'entorn i les circumstàncies. La web es torna més intel·ligent i autònoma.

Selectors

Un selector és una estructura condicional en què hi ha diverses possibilitats. No sempre és blanc o negre, de vegades tenim tonalitats de gris o colors. De fet, els condicionals de què parlàvem abans no són més que selectors duals.

JavaScript utilitza els selectors per dirigir el flux de la pàgina en diverses direccions a partir del valor que conté una variable. Si, per exemple, hem fet una web en diversos idiomes, podem utilitzar un selector per encaminar l'usuari cap a l'opció escollida.

En aquest mòdul:

Pràctica 1: **Bucles for**
Pràctica 2: **Recórrer matrius i objectes**
Pràctica 3: **Bucles while**
Pràctica 4: **Condicionals i selectors**

Exercicis: **Menús d'enllaços**

Per fer les pràctiques, obrim el Dreamweaver i ens situem a la "Vista de código", al mig de la secció <body> que, normalment, és a la línia 8. Allí hem d'inserir el codi dels exercicis. Si no tenim activada la numeració de línies, anem al menú *Ver - Opciones de vista de Código - Números de líneas* i l'activem.

En el disseny de pàgines sovint hem de repetir la mateixa estructura moltes vegades, amb lleugeres variacions de contingut o format. Copiem el bocí que ens interessa i l'enganxem en un altre lloc o, si volem repetir l'estructura en pàgines diferents, utilitzem plantilles.

Els bucles permeten escriure fragments de codi només un cop i repetir-los mentre es compleixi una condició, tot inserint-hi elements variables. Són al nucli de tots els llenguatges de programació i constitueixen una eina molt poderosa. En aquesta pràctica proposem quatre exercicis per entendre els **bucles for**, un dels recursos més usats en JavaScript.

Al primer mòdul ja vam tenir la primera experiència amb el llenguatge i, si hem arribat fins aquí sense errades, ens podem sentir afortunats. Escriure codi és, bàsicament, cometre errades. Bé, potser no cal exagerar, però, per si un cas, en acabar aquesta pràctica hi ha uns consells útils. (Si no podeu esperar, feu **clic aquí**)

Conceptes JavaScript en aquest capítol

Declaració **for() :** determina quantes vegades s'han de repetir les instruccions.
Declaració **var :** declara una variable i li assigna un valor.
Declaració **if() :** condicional que avalua si una expressió o comparació és certa.
Mètodes **document.write() , document.writeln() :** escriuen a la pàgina.
Assignació **= :** estableix el valor d'una variable.
Comparacions **== , < , >= :**igual, menor que, major o igual que.
Operacions **++ , + , - , * , % :** increment, suma, resta, multiplicació i residu de la divisió.

Fixeu-vos: utilitzem "**=**" per assignar i "**==**" per comparar. Confondre-les pot resultar fatal.

1.- Comptar fins a 100

Els bucles **for** es basen en una **variable** numèrica que fa d'índex. Comença per un valor, estableix una condició per mantenir-se funcionant i defineix una operació per assignar nous valors a la variable. Comptar fins a 100, seria, en el nostre llenguatge:

- *Començar per 1 i, mentre el número sigui menor que 101, incrementar un.*

La declaració del bucle **for** té tres elements, entre parèntesi i separats per punt i coma:
- Inicialització de la variable i assignació del primer valor: **var i=1**
- Condició per mantenir-se dins del bucle: **i<101**
- Incrementar un: **i++**

La declaració queda, doncs: **for (var i=1 ; i<101 ; i++)**

En cada volta s'executen les instruccions que hi ha entre les claus **{ }**. Normalment, aquest bocí del codi s'escriu tabulat (més a la dreta) per tal de facilitar la lectura. Aquest és un hàbit que cal adquirir. En estructures més complexes o scripts més llargs, resulta imprescindible.

En el cas que ens ocupa, el que fem és, senzillament, escriure a la pàgina el valor que té la variable, seguit d'un espai en blanc: **document.write(i + ' ')**

```
<script language="javascript">
<!--
for ( var i=1 ; i<101 ; i++ ) {
  document.write( i + ' ' ) ;
}
// -->
</script>
```

2.- Donar format

L'exemple anterior fa el que volíem, però el resultat és lineal. Cal donar format. Per això, inserim el codi JavaScript dins d'una taula. La farem amb "borde", alineada al centre. Els números s'escriuran en cel·les, alineats a la dreta, i cada 10, començarà una nova fila.

Utilitzem l'operador %, que dóna com a resultat el mòdul (residu) de la divisió, per saber quan hem de començar i acabar la fila (obrir i tancar <td>): si el mòdul de dividir l'índex per 10 és 1, obrim la fila, i si és 0, la tanquem.

La funció **if** executa el codi que s'escriu a continuació si és veritat l'expressió que avalua. Si i%10 és 1, escriu un <tr>. **document** és la representació interna de la pàgina - un objecte - que maneja el navegador i **write** és el mètode per escriure-hi. Així, doncs:

- quan escrivim **if (i%10==1) document.write('<tr>')**
- estem dient: *si el residu de dividir per 10 és 1 escriu a la pàgina "<tr>"*

A la línia central del bucle, que s'executarà en totes les passades, utilitzem els dos tipus de cometes. Les simples (apòstrof) delimiten el text que s'envia al document. Les dobles formen part d'aquest text, que no és més que llenguatge HTML. La variable (**i**), evidentment, queda fora de qualsevol tipus de cometes.

```
<table border="1" cellspacing="0" align="center">
<script language="javascript">
<!--
for ( var i=1 ; i<101 ; i++ ) {
  if ( i%10==1 ) document.write( '<tr>') ;
  document.write( '<td width="30" align="right">' + i + '</td>' ) ;
  if ( i%10==0) document.write( '</tr>' ) ;
}
// -->
</script>
</table>
```

3.- Ordre Invers, només parells, deu a cada fila

Un bucle també pot anar decreixent, i no cal que ho faci d'un en un. El tercer element de la declaració pot ser qualsevol operació vàlida, en aquest cas, restar-ne dos: **i = i - 2**

Mantindrem el bucle mentre sigui més gran o igual que 0. Hem d'ajustar les condicions d'inici i fi de cada fila. Tornem a utilitzar l'operador mòdul, **%**, ara dividint per vint. Comencem pel 98. El primer residu és, doncs,18, i l'últim, 0. Aquest cop, a més, pintem les cel·les.

for(), **if()**, **write()**, ... tenen forma de funció (veurem les funcions més endavant). Pràcticament totes les accions en JavaScript utilitzen aquest format**:** instrucció seguida de parèntesis i, a dins, els ***arguments***, o paràmetres, sobre els quals s'executa l'acció. Aquests arguments poden ser variables, codi HTML, altres funcions, ...

```
<table border="0" cellspacing="1" align="center">
<script language="javascript">
<!--
for ( var i=98 ; i>=0 ; i=i-2 ) {
  if ( i%20==18 ) document.write( '<tr>') ;
  document.write( '<td align="center" bgcolor="#FFCC99" >' + i + '</td>' ) ;
  if ( i%20==0) document.write( '</tr>' ) ;
}
// -->
</script>
</table>
```

4.- Bucles enniuats: les taules de multiplicar

Podem fer un bucle dins d'un altre, cadascun amb la seva variable-índex. D'aquesta manera, en cada volta del bucle exterior s'executa tot el bucle interior.

El nostre sistema de numeració es basa en bucles enniuats. Penseu en un comptador, per exemple, el de la llum. Les xifres de les unitats són el bucle més interior. Giren i, després del 9, fan saltar la xifra de les desenes (bucle extern al de les unitats), i tornen a començar per 0. El mateix passa entre desenes i centenes. Imagineu fins on pot arribar el nivell

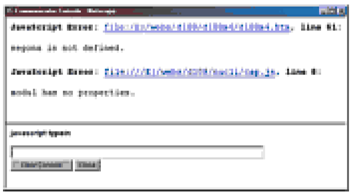
d'enniuament.

En aquest cas, les files de la taula s'inicien i s'acaben al bucle de la variable "**i**", i les cel·les, al de la variable "**x**". Utilitzem el mètode **writeln** per tal de forçar un salt de línia en acabar. El mètode **write** no ho fa i, en conseqüència, genera un codi font més difícil de llegir.

```
<table border="1" cellspacing="0" align="center">
<script language="javascript">
for ( var i=1 ; i<11 ; i++ ) {
  document.writeln( '<tr>')
  for ( var x=1 ; x<11 ; x++ ) {
    document.writeln( '<td width="30" align="center">' + (i*x) + '</td>' )
  }
  document.writeln( '</tr>' )
}
</script>
</table>
```

A l'exemple anterior no hem utilitzat els signes d'ocultar el codi als navegadors anteriors a la versió 4, **<!--** per començar i **//-->** per acabar. Tampoc hem usat el **;** per acabar les línies. Si tenim navegadors actuals, funciona, si no ... què hi fem en un curs de JavaScript?

5.- Com localitzar les errades



Quan escrivim codi Javascript és normal cometre errades. Sortosament, el propi navegador ens ajuda a localitzar-les. Netscape ho posa fàcil: esborrem la barra d'adreces i escrivim: ***javascript:*** (atenció als **:** finals). Premem Intro i s'obrirà una finestra que ens informa.

Internet Explorer mostra un triangle groc i una descripció a l'esquerra de la barra d'estat. Si fem doble clic sobre el triangle, s'obrirà una finestra amb més informació. I, encara més fàcil, el podem configurar per tal que la finestra s'obri tota sola:

1.- Anem el menú *Herramientas, Opciones de Internet, Opciones Avanzadas*.
2.- Marquem *Mostrar una notificación sobre cada error de secuencia de comandos*.
3.- Acceptem. Quan aparegui l'errada, potser haurem de fer clic *Mostrar detalles*.

Si el nostre navegador i/o sistema operatiu disposen d'un depurador de codi, no cal utilitzar-lo, de moment. Serà més ràpid i còmode fer-ho a mà. Si utilitzem l'Explorer i el tenim activat, tornem a les *Opciones Avanzadas* i marquem *Deshabilitar depuración de secuencias de comandos*.

Una de les errades freqüents es produeix per manca de "paritat" entre els signes (cometes, parèntesis). Si, per exemple, escrivim una frase amb un apòstrof entre cometes simples ('), s'interpretarà l'apòstrof com a fi de cadena. Es pot solucionar escrivint una antibarra davant l'apòstrof (\'). Aquesta tècnica farà que s'interpreti com a part de la cadena.

Tots dos navegadors ens diuen el **tipus** d'errada i la **línia** on s'ha produït, a partir de l'inici de l'arxiu, de manera que és fàcil localitzar-la. El problema és que, sovint, una errada en provoca una altra, i la llista pot ser llarga. En aquest cas, hem de localitzar i arreglar primer la primera errada, i provar. Potser les altres hauran desaparegut.

Si utilitzem Dreamweaver com a editor, ja devem saber que disposa d'una excel·lent eina., un depurador de codi JavaScript en temps de disseny. S'explica a la propera pràctica.

Estructures JavaScript > **Recórrer matrius i objectes**

A la pràctica anterior hem treballat amb números. Està bé per entendre com funcionen els bucles. La seva utilitat, però, no es limita a fer operacions aritmètiques i escriure'n el resultat. En aquesta pràctica veurem com es treballa amb matrius i objectes.

Una matriu (**Array**) és una col·lecció de dades indexades, per exemple, una llista. És un objecte predefinit de JavaScript que té elements, propietats i mètodes.

- Els **elements** són els valors de la llista.
- Les **propietats** són les característiques d'aquests elements o de la pròpia llista.
- Els **mètodes** són les accions que es poden executar.

Si pensem en l'Array "Dies de la setmana", un element seria "dijous", una propietat d'aquest element seria "està format per lletres", una propietat de la llista seria el nombre d'elements (7) i un mètode seria "escriure la llista en ordre invers".

Un objecte (**object**) té una definició una mica més àmplia i abstracta, però similar. Pensem, per exemple, en el concepte "bombeta". Coneixem les seves propietats (potència, forma, color, ...) i mètodes (s'encén, s'apaga, es fon, ...). Sabem tot això, en general, però només ho podem aplicar a bombetes concretes, no pas al concepte.

Pel que fa als llenguatges de programació, el concepte s'anomena "objecte" i la realitat concreta, "instància". En el cas d'abans, "matriu (Array)" era l'objecte i "dies de la setmana", la instància que hem triat. I, al paràgraf anterior, podríem dir que "bombeta" és un objecte i "la meva bombeta de 60w marca Acme" és la nostra instància concreta.

Sovint, però, s'utilitza la paraula "objecte" per referir-se a una instància, igual com diem "el cotxe" per referir-nos a un vehicle en particular. JavaScript no fa distincions.

El navegador crea una jerarquia d'objectes per tal de poder manejar la pàgina i JavaScript pot manipular aquesta estructura. Els enllaços, per exemple, són objectes d'aquesta jerarquia, com ho són les imatges, els estils, els formularis, la finestra o el mateix navegador.

Conceptes JavaScript en aquest capítol

Constructor **new Array() :** crea una instància de matriu i, opcionalment, li assigna valors.
Propietat **lenght:** numèrica, indica la longitud (nombre d'elements) d'un objecte o Array.
Declaració **for ... in :** bucle per recórrer un objecte.
Constructor **new String() :** crea una instància de cadena i li assigna un valor.
Propietat **href :** informació sobre l'adreça.
Mètode **indexOf() :** retorna la posició d'un caràcter o una subcadena en una cadena.

Atenció: Hem d'utilitzar majúscules i minúscules tal com s'han escrit aquí.

1.- Els dies de la setmana

Utilitzarem l'exemple del principi. Farem la llista i l'escriurem en una taula amb l'ajut d'un bucle. Per això, declarem la variable dies i li assignem un Array:: **var dies = new Array(...)**.

Dins dels parèntesis escrivim els elements entre cometes, separats per comes (després de l'últim, no). Ho podem fer en una mateixa línia, però, si ho fem com a l'exemple, es millora la llegibilitat. Quan ens volem referir a un element utilitzem la sintaxi **element [índex]**. El primer element té l'índex 0, no 1. La propietat **lenght** ens indica quants elements hi ha.

Un cop acabada la declaració, fem un bucle que llegeixi l'Array des de la posició 0 (dilluns) i mentre l'índex sigui menor que la longitud (lenght, 7). Cada volta del bucle crea una fila amb una cel·la de 140 píxels, alineada al centre, que conté un element de l'Array.

```
<table border="1" cellspacing="0" bgcolor="#CCDDEE">
<script language="javascript">
<!--
var dies = new Array(
"Dilluns",
"Dimarts",
"Dimecres",
"Dijous",
"Divendres",
"Dissabte",
"Diumenge" ) ;
for( var i=0; i<dies.length; i++ ){
   document.writeln('<tr><td width="140" align="center">' + dies[i] + '</td></tr>') ;
}
// -->
</script>
</table>
```

2.- Llista d'enllaços

JavaScript pot manejar Arrays de més d'una dimensió. De fet, cada element d'un Array pot ser, al seu torn, un altre Array. Imaginem-nos fins on podem arribar. L'exemple més senzill seria un bidimensional, que el podem imaginar situat en una graella o un full de càlcul.

Per fer una llista d'enllaços, però, en tenim prou amb tractar l'Array com a parelles, és a dir, incrementar el comptador del bucle de dues en dues (o de tres en tres, si, per exemple, volem que cada enllaç tingui una imatge diferent).

Això sí, és necessari que les parelles segueixin un ordre i, al bucle, que cada element del parell vagi on ha d'anar. Per tal de visualitzar-ho millor, utilitzarem dues variables dins del bucle, **url** i **nom**, tot i que no caldria.

```
<table border="1" cellspacing="0" bgcolor="#DDEECC">
<script language="javascript">
var lnk = new Array(
"http://www.xtec.es",       "Web de la Xtec",
"http://www.edu365.com",    "edu365.com",
"http://www.educalia.org",  "Educàlia",
"http://www.google.com",    "Google" )
for( var i=0; i<lnk.length; i=i+2 ){
  var url = lnk[i]
  var nom = lnk[i+1]
  document.writeln('<tr><td width="140" align="center">')
  document.writeln('<a href="'+url+'">'+nom+'</a></td></tr>')
}
</script>
</table >
```

3.- Mostrar els enllaços d'una pàgina

Ara plantegem la qüestió gairebé oposada a l'anterior, en lloc de fer una llista d'enllaços, comprovarem els que hi ha en una pàgina. Els enllaços, com hem dit abans, són objectes dins del document. Farem un bucle que els mostrarà.

Utilitzarem el bucle **for ... in**, una variant dels bucles for, ideal per recórrer objectes. La seva sintaxi és més senzilla: **for** (variable **in** objecte). En cada volta la variable pren el valor de l'objecte recorregut, de manera que ens hi podem referir fàcilment.

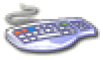
El navegador coneix l'objecte **document.links** de la pàgina i ho aprofitarem. La sintaxi dels objectes es basa en escriure la jerarquia de dalt a baix, o de general a particular, amb els noms separats per punts. Quan un dels elements és un altre objecte, utilitzem els claudàtors per assenyalar l'índex, com fèiem als Arrays.

Com que, d'entrada, no sabem quin tipus de dades ens proporcionarà el recorregut, a cada volta crearem una variable, url, i li assignarem el valor que ens proporcioni l'objecte, convertit en cadena. "Cadena" significa literal, format per lletres.

- La línia
  `var url = new String( document.links[item].href )`

- Vol dir.
la variable **url** serà una nova cadena, amb el valor que té la propietat **href** de l'objecte **links** de l'**item** que està passant en aquesta volta, i que pertany a **document** (la pàgina actual).

Un cop fet això, comprovem si **url** conté el text ".htm", per saber si li hem de posar un enllaç. Per això, utilitzem el mètode **url.indexOf()**, que ens informa sobre la posició que ocupa una subcadena en una cadena. Si és major que 0, aleshores és un enllaç. L'escrivim al document, preparat per obrir-se en una finestra nova (target="_blank").

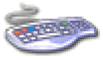


```
<script language="javascript">
for( var item in document.links ){
  var url = new String( document.links[item].href )
  if( url.indexOf(".htm")>0 ){
    document.writeln('<a href="' + url + '" target="_blank">' + url + '<br>')
  }
}
</script>
```

4.- La sintaxi del punt

Com hem vist, la sintaxi per accedir a les propietats dels objectes és relativament senzilla, es basa en separar la jerarquia amb punts. Per executar els seus mètodes hem d'afegir, a més, el parell de parèntesis i, a dins, els paràmetres, si cal.

Sense aprofundir més en el tema, veurem algunes propietats d'objectes. Quins? Intentem esbrinar-ho ... Més endavant hi tornarem.



```
<script language="javascript">
<!--
document.writeln( document.title + '<br>');
document.writeln( document.lastModified + '<br>');
document.writeln( navigator.appName + '<br>');
document.writeln( window.screen.width + 'x' + window.screen.height + '<br>');
document.writeln( document.forms[0].method + '<br>');
document.writeln( document.images[1].src + '<br>');
// -->
</script>
```

5.- Depuració d'errades amb Dreamweaver

Si utilitzem Dreamweaver com a editor, disposem d'un depurador de codi JavaScript que ens permet detectar les errades en temps de disseny (bé, no és del tot cert, ja que obliga a guardar l'arxiu, però el codi queda editable).

Per iniciar la depuració obrim el menú "Archivo - Depurar en navegador" o fem clic a la barra d'eines el botó "Vista previa / Depurar en navegador" i triem el que ens calgui.

Si hi ha errades de sintaxi, apareix la finestra "Errores de sintaxi JavaScript". Quan ens situem sobre un element de la llista, mostra l'explicació al quadre inferior i un botó "Ir a línea" que ens permet anar al punt del codi que conté l'errada.

Si el depurador localitza errades de lògica, en canvi, apareix la finestra "Depurador JavaScript", que ens permet, per exemple, definir punts d'aturada, executar el codi línia a línia, veure l'estat de les variables, ... Però, tot això i més ho tenim a l'ajuda del Dreamweaver, a la secció "Depurar código JavaScript".

Estructures JavaScript > **Bucles while**

En la pràctica 1 d'aquest mòdul hem estudiat una estructura de control repetitiva: els bucles **for**. Aquesta estructura té la particularitat que el nombre d'iteracions a efectuar és un valor constant i que s'ha de conèixer des d'un principi.

Hi ha altres ocasions en què també s'ha d'aplicar estructures de control iteratives, però, sense saber inicialment, quin serà el nombre de repeticions a efectuar. Per a aquests casos existeix l'estructura de control **while**.

La funció d'un bucle *while* és repetir un bloc accions mentre sigui certa una determinada condició.

Evidentment, la condició s'haurà de modificar dins de les accions a repetir i en alguna ocasió futura haurà de canviar el seu valor a fals, en cas contrari, si la condició mai canviés, es produiria un bucle sense fi que provocaria un error del programa.

En aquesta pràctica veurem la utilització de la sentència **while**, de la seva variant **do ... while** i altres conceptes associats, conjuntament amb exemples senzills d'aplicació.

Elements JavaScript en aquest capítol.

Sentència **while() :** executa un bloc de codi mentre una condició sigui certa, és possible que no s'executi cap vegada..
Sentència **do ... while() :** executa un bloc de codi mentre una condició sigui certa, almenys una vegada.
Sentència **continue :** provoca que es torni a executar des de l'inici el bloc de codi.
Sentència **break :** provoca que finalitzi l'execució repetitiva del bloc de codi, independentment del valor de la condició.
Mètode **parseInt(cadena) :** Converteix en valor numèric enter el contingut de cadena. Si no es pot, retorna NaN (Not a Number).
Mètode **isNaN(valor) :** Retorna true si el contingut de valor no és numèric, en cas contrari, retorna false.

1.- La mitjana aritmètica. Bucles while().

La sintaxi genèrica de **while** és la següent:

```
while(condició)
{
  ..instruccions..
  [break]
  ..instruccions..
  [continue]
  ..instruccions..
}
```

Aquesta estructura de control s'ha de llegir de la següent forma:
Mentre la condició sigui certa fer ...

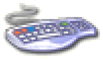
Els elements que estan entre [ ] són d'ús opcional i es descriuran més endavant. El bloc d'instruccions que es trobi entre { i } s'executarà de forma repetitiva mentre el resultat de **condicio** sigui **true**.

La condició pot estar formada per una variable booleana (que sol pot contenir dos valors **true** o **false**) o bé per una operació lògica o de comparació que retorna com a resultat un valor booleà.

L'exemple següent ens mostra el funcionament d'aquesta estructura de control. L'objectiu és calcular la mitjana d'un conjunt de números. Aquests números s'introduiran per part de l'usuari mitjançant un quadre de diàleg. Inicialment es desconeix el nombre de valors que

s'introduiran.

La introducció dels números finalitzarà quan l'usuari premi sobre el botó "Cancelar" del quadre de diàleg, per tant, inicialment no sabem quina quantitat de números s'introduiran.



```
<script language="JavaScript">
  <!--
  var acumulat=0;
  var nValors=0;
  var valor=prompt("Introdueix un no. (CANCELAR per acabar)","");
  while (valor != null)
  {
    acumulat += parseInt(valor);
    ++nValors;
    valor=prompt("Introdueix un no. (CANCELAR per acabar)","");
  }
  alert("La mitjana val: " + eval(acumulat/nValors));
  // -->
</script>
```

En el funcionament d'aquest codi, es declaren i s'inicialitzen les variables *acumulat*, *nValors* i *valor*. Les dues primeres, seran de tipus numèric, ja que s'han inicialitzat amb zero, ara bé, la variable *valor* s'inicialitza a un format de cadena de text, ja que la funció *prompt* retorna una cadena de text.

La variable **acumulat** contindrà la suma acumulada dels valors numèrics i **nValors** la quantitat de valors introduïts per l'usuari.

La línia *while (valor != null)* és l'inici del bloc *while* i el contingut del parèntesi la condició que ha de ser *certa* perquè es repeteixi el contingut interior del bucle. S'ha de tenir en compte que el quadre de diàleg **prompt** retorna el valor **null** quan es prem a sobre del botó *Cancelar*. Aquesta condició compara el contingut de la variable de cadena *valor* amb *null*. Si són diferents retornarà el valor booleà *true*, en cas contrari retornarà *false*.

A l'interior del bloc *while*, tenim les següents línies de codi:

    acumulat += parseInt(valor);

Aquesta línia afegeix a la variable *acumulat* el valor numèric de la variable *valor*, l'operador d'assignació += equival a: *acumulat = acumulat + parseInt(valor);*. La funció *parseInt()* converteix la cadena de text que conté *valor* a un valor numèric, per poder-ho sumar a la variable *acumulat*.

    ++nValors;

Aquesta línia incrementa en una unitat a la variable *nValors*, l'operador aritmètic **++** equival a la següent operació: *nValors = nValors + 1*.

    valor=prompt("Introdueix un no. (CANCELAR per acabar)","");

Aquesta última línia del bucle és igual a la que existeix just abans d'iniciar el bucle *while*. La seva funció és sol·licitar un nou valor a l'usuari i assignar-lo a la variable *valor*. L'existència d'aquesta línia és molt important ja que introdueix la variació sobre la variable que s'avalua en la condició de *while*, si la variable *valor* no canvies mai, no seria possible acabar el bucle *while* i, generalment, produiria un error.

En el moment que l'usuari prem el botó *Cancelar*, la condició del bucle *while* passa a ser *false* i finalitza el bucle de repetició. S'ha de remarcar que hi ha la possibilitat que el contingut del bucle *while* no s'arribi a executar ni una sola vegada, això es pot aconseguir si en l'execució del primer *prompt*, que està just abans de l'inici del *while*, es prem directament sobre *Cancelar*.

Finalment, quan s'ha sortit del bucle, s'executa la línia:

    alert("La mitjana val: " + eval(acumulat/nValors));

Aquesta línia mostra un missatge amb el text *"La mitjana val: "* concatenat amb el resultat d'avaluar l'operació algebraica: *acumulat / nValors* que ens dóna la mitjana aritmètica del conjunt numèric introduït per l'usuari.

2.- Evitem els errors.

El programa de l'apartat 1 és molt vulnerable. Suposem que l'usuari introdueix un valor que no és numèric o en blanc, en aquest cas, es continuarà de forma normal, però, a l'hora de mostrar el resultat veurem que no és correcte (mostrarà NaN, Not a Number).

També hi cap la possibilitat que l'usuari premi *Cancelar* just en la primera consulta, en aquest cas ja s'ha dit que el bloc del bucle no s'arribarà a executar mai, això provoca que la variable *nValors* contingui el valor 0 i a la línia de l' *alert* es farà una divisió per zero ja que aquesta variable està situada en el denominador. S'ha d'evitar que en aquesta situació es faci la divisió per zero.

Les instruccions *break* i *continue* es poden introduir dins l'estructura *while* (també dins el *for* i el *do...while*) i provoquen un comportament diferent d'aquesta estructura. Habitualment se solen executar en funció d'una determinada situació que es detecta mitjançant una estructura condicional **if**.

La instrucció **break** situada dins un bucle, provoca la interrupció de l'execució de les instruccions del bloc i finalitza el bucle, independentment del resultat de la condició del bucle.

La instrucció **continue**, en canvi, provoca que es torni a iniciar l'execució de les instruccions del bucle sense acabar la seqüència actual.

L'exemple següent resol els inconvenients indicats prèviament. És a dir, quan s'introdueixi un valor que no pugui ser convertit en numero enter o sigui un camp buit, s'executarà un *break* que provocarà la finalització del bucle independentment del valor de la condició del bucle. per fer-ho es fa anar la línia:

```
if(isNaN(valor) || valor=="") break;
```

if és un condicional que s'estudia amb més detall en la pràctica següent. Dins el parèntesi tenim una condició que avalua si el contingut de la variable *valor* és numèrica mitjançant el mètode **isNaN(),** que verifica si el contingut de *valor* pot ser convertit en número. També s'avalua si el contingut de *valor* és un valor buit (*valor==""*). Si es compleix una condició o l'altra se surt del bucle amb el **break**. El símbol **||** indica l'operació lògica **o**.

Si desitgéssim menysprear el valor no numèric i tornar a demanar un nou valor sense sortir del bucle, podríem utilitzar la sentència **continue** per tornar a iniciar el bucle, però abans, hauríem de tornar a demanar un nou valor numèric a l'usuari, advertint-lo del seu error. La línia que conté el *break* la podríem substituir per les següents:

```
if (isNaN(valor) || valor=="") {
   valor=prompt("No. erroni, introdueix un nou valor numèric (FI per acabar)","");
   continue;
}
```

En el cas d'utilitzar l'opció amb la sentència **continue**, si s'introdueix un valor no numèric es tornarà a demanar a l'usuari el nou valor fins que aquest sigui correcte o bé, premi sobre Cancelar.
per evitar l'error de la divisió per zero verificarem si **nValor** no val zero, si és *cert* no s'executarà la instrucció **alert**, les següents línies de codi serveixen per aquesta verificació:

```
if (nValors != 0) alert("La mitjana val: " + eval(acumulat/nValors));
else alert("Ep! No has introduït cap valor");
```

El codi complet és el que s'indica a continuació, s'hi ha utilitzat l'opció de *break* en el cas que s'hagi introduït un valor no numèric. Podeu fer la prova de substituir la línia del *break* per les línies de codi indicades prèviament amb l'opció de *continue* i comprovar el seu funcionament diferenciat.

```
<script language="JavaScript">
  <!--
  var acumulat=0;
  var nValors=0;
  var valor=prompt("Introdueix un no. (CANCELAR per acabar)","");
  while (valor != null)
```

```
  {
    if(isNaN(valor) || valor=="") break;
    acumulat += parseInt(valor);
    ++nValors;
    valor=prompt("Introdueix un no. (CANCELAR per acabar)","");
  }
  if (nValors != 0) alert("La mitjana val: " + eval(acumulat/nValors));
  else alert("Ep! No has introduït cap valor");
  // -->
</script>
```

3.- Llista de nombres parells. Bucles do ... while().

L'estructura iterativa **do ... while** és molt semblant al comportament de *while*. L'única diferència és que en aquest cas, la *condició* s'avalua al final del bloc, per tant, a diferència del *while,* el bloc de codi a repetir s'executarà sempre, com a mínim una vegada.

La sintaxi genèrica d'aquesta estructura és:

```
do {
  ..instruccions..
  [break]
  ..instruccions..
  [continue]
  ..instruccions..
} while(condicio)
```

Aquesta estructura de control s'ha de llegir de la següent forma:
fer ... mentre la condició sigui certa.

El següent exemple escriu sobre el document els nombres parells entre 0 i el màxim indicat per l'usuari. Per fer-ho s'utilitza la sentència **do ... while**. Observeu com, encara que es premi *Cancelar* o *Aceptar* amb un valor zero o buit o no numèric, sempre apareix com a mínim el primer valor 0.

```
<script language="JavaScript">
  <!--
  var fi=prompt("Introdueix el valor màxim:","");
  var valor=0;
  do {
      document.writeln(valor + " ");
      valor = valor + 2;
  } while (valor < fi)
  // -->
</script>
```

4.- Els números primers.

L'exemple que tenim a continuació mostra una altra aplicació amb l'estructura de control *while*. Aquest és una mica més complex, ja que el bloc contingut dins el *while* és més extens i conté altres estructures de control.

L'objectiu d'aquest codi consisteix en escriure sobre la finestra un nombre determinat de nombres primers començant pel 1.

Inicialment ens sol·licita la introducció del màxim nombre primer a representar mitjançant una finestra de diàleg emergent, aquest valor el desa a la variable **fi**.

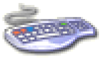
Posteriorment s'inicia el bucle **while** que s'executarà de forma iterativa mentre la variable **numero** tingui un valor inferior al valor contingut per la variable **fi**. Al final del bucle *while* tenim la línia amb el codi ++numero; encarregada d'incrementar el valor de *numero* i a la vegada, modificar el valor de la condició.

També dins el bucle *while* hi tenim una altra estructura iterativa, en aquest cas és la **for**. Aquest bucle *for* s'encarrega de determinar si *numero* és un valor primer o no. Per fer-ho, s'utilitza la variable *primer* com a booleana i se li assigna inicialment el valor *true.*

Dins el bucle **for**, s'efectua múltiples divisions entre el **numero** i qualsevol altre número entre 2 i *numero/2*. A cada divisió es verifica si el **residu** (%) és igual a 0, en cas de ser-ho, vol dir que el *numero* no és primer i es posa la variable *primer* a *false* i sortim del bucle *for* mitjançant el *break*.

Seguidament, si **primer** val **true**, vol dir que el *numero* és primer i s'escriu sobre la finestra activa el valor de *numero*.

El bucle *while* es repetirà tants cops com sigui necessari fins arribar al número màxim indicat per l'usuari, desat dins la variable **fi**.



```
<script languaje="JavaScript">
//numeros primers.

  var fi=prompt("Introdueix el màxim nombre primer a visualitzar:","");
  var a, primer, numero=1;
  while(numero <= fi) {
      primer = true;
      for(a=2; a<=numero/2; a++) {
          if ((numero % a) == 0) {
              primer = false;
              break;
          }
      }
      if (primer) document.write(numero + " ");
      ++numero;
  }
</script>
```

Les pàgines es llegeixen i s'executen de forma seqüencial, de dalt a baix i d'esquerra a dreta. HTML no té cap més possibilitat. Si utilitzem capes i fulls d'estils, podem situar elements a qualsevol lloc de la pàgina, però, com abans, tot està decidit.

Amb JavaScript, en canvi, podem dirigir el flux en funció de què es compleixin, o no, unes condicions. Ja ho hem vist en pràctiques anteriors. Ara hi tornem amb més profunditat.

Un **condicional** fa una comparació o, si ho volem més precís, "avalua una expressió". Si el resultat és cert (true), s'executa una instrucció, un bloc de codi o tota una pàgina, el que vulguem. Opcionalment, si el resultat és fals (false) es pot tirar per un altre camí.

Sovint, però, dues opcions són poques. L'expressió que s'avalua pot tenir diversos resultats i cal un mecanisme que permeti triar sense haver de fer eleccions binàries. Els **selectors** ens ofereixen, precisament, aquesta funcionalitat.

Conceptes JavaScript en aquest capítol

Declaració **if() ... else :** avalua una expressió i dirigeix el flux segons el resultat.
Operador **? :** avalua una expressió i retorna (assigna) un valor segons el resultat.
Mètode **confirm() :** presenta un missatge i permet triar entre dues respostes.
Declaració **switch() ... case :** estructura que permet seleccionar entre diverses opcions.
Objecte **Math :** proporciona una interfície per treballar amb números.
Mètode **random() :** genera un número aleatori entre 0 i 1.
Mètode **floor() :** elimina els decimals d'un número.
Mètode location**.reload() :** torna a carregar la pàgina.

1.- Paper de llistat

Recordeu el paper d'impressora a ratlles blanques i blaves? De vegades, va bé tenir colors alterns, sobretot, quan visualitzem llistats de dades. Veurem com es pot fer.

Necessitem un interruptor, alguna cosa que només tingui dues possibilitats. Alguna idea? Naturalment, el mòdul de dividir per 2, parells i senars. Tornem a utilitzar una taula i pintem les cel·les. Al mateix temps, veurem com es poden escriure instruccions dinàmiques, a bocinets, en funció d'una decisió.

"*Si es compleix la condició, per aquí, si no, per allà*", en JavaScript seria:

```
if ( condició ) {
    ... bloc cert ...
}
else {
    ... bloc fals ...
}
```

Al codi, utilitzem el mètode **document.write()** quan no volem fer un salt de línia, és a dir, quan volem que el text d'una instrucció s'escrigui a la pàgina "enganxat" al següent. En canvi, utilitzem el mètode **document.writeln()** per fer un salt de línia després d'escriure. Cal recordar que HTML reconeix els salts de línia i que és HTML el que escrivim al document.

Aplicarem els colors a les files (<tr>), no pas a les cel·les (<td>). Això també ho podem fer amb Dreamweaver, si sabem usar la barra d'estat, tal com s'explica al mòdul 1. Cada fila, tindrà dues cel·les del mateix color. La de l'esquerra ens servirà per numerar les línies.

```
<table border="0" cellspacing="0" width="100%">
<script language="javascript">
<!--
for ( var i=1 ; i<21 ; i++ ) {
  document.write( '<tr bgcolor="#' ) ;
```

```
  if ( i%2==1 ){
    document.write( 'FFFFFF') ;
  }
  else {
    document.write( 'DDEEFF') ;
  }
  document.writeln( '"><td width="20" align="right">' + i + '</td>' ) ;
  document.writeln( '<td> </td></tr>' ) ;
}
// -->
</script>
</table>
```

2.- Més compacte

Per fer això, però, no cal tant codi. JavaScript disposa d'un **operador ternari** (és l'únic que té tres elements), que pot resoldre aquesta situació de forma més compacta i elegant. És un "condicional en línia" que actua com un "if … else …", però de manera més breu:

(condició **)** **?** bloc cert **:** bloc fals

Si es compleix la condició s'executa el primer bloc, si no, el segon. Això resulta molt útil, sobretot quan tots dos, cert i fals, són curts o només s'ha de retornar un valor.

L'expressió: **(i%2==1) ? 'EEFFDD' : 'EEDDFF'**
es llegeix: *Si el residu de dividir* **i** *per* **2** *és* **1**, *retorna* 'EEFFDD', *si no, retorna* 'EEDDFF'

```
<table border="0" cellspacing="0" width="100%">
<script language="javascript">
for ( var i=1 ; i<21 ; i++ ) {
  document.writeln( '<tr bgcolor="#' + ( ( i%2==1 ) ? 'EEFFDD' : 'EEDDFF' ) + '">' )
  document.writeln( '<td width="20" align="right">' + i + '</td>' )
  document.writeln( '<td> </td></tr>' )
}
</script>
</table>
```

3.- Tu tries

Als exemples anteriors, l'usuari fa d'espectador. Si li volem donar l'oportunitat de triar, necessitem un mecanisme que retorni cert o fals. El tenim. Es tracta de la funció "confirm", germana d'*alert* i cosina de *prompt*, que hem vist al mòdul anterior. Tots tres són mètodes de l'objecte **window**. Si no s'especifica la finestra, es suposa l'actual.

La seva sintaxi és molt senzilla: **confirm (** missatge **)** Es pot usar de dues maneres:

1.- Assignem el resultat a una variable: var tu_tries = confirm (missatge)
2.- Avaluem el resultat a la pròpia estructura condicional:

```
if ( confirm ( missatge ) ) { … resposta afirmativa … }
else { … resposta negativa … }
```

Com es pot veure, és una forma ràpida i eficaç de consultar l'usuari. Això sí, només té dues opcions i obliga a redactar de manera que s'hagi de triar entre Acceptar o Cancel·lar.

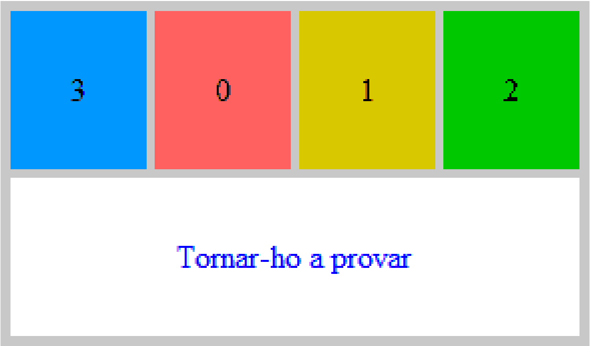
```
<script language="javascript">
if ( confirm ( 'Vols veure una bona imatge?' ) ) {
  document.writeln( '<img src="images/foto01.jpg">' )
}
else {
  document.writeln( 'Tu t'ho perds !' )
}
</script>
```

4.- Joc d'atzar

Un selector (**switch**) és una estructura que permet dirigir el flux del programa en diverses direccions, depenent del valor d'una variable. Cada cas (**case**) executa un bloc d'instruccions. Si aquest bloc es tanca amb la instrucció de ruptura (**break**), salta a l'acabament de l'estructura; si no, llegeix el cas següent.

Opcionalment, podem utilitzar la paraula **default** per indicar què s'ha de fer si no s'ha donat cap dels casos previstos. Això només té sentit si els blocs anteriors acabaven amb "break", en cas contrari, sempre s'executaria el bloc de "default".

```
switch ( expressió ) {
  case cas1 :
    bloc d'instruccions per al cas1;
    break;
  case cas2 :
    bloc d'instruccions per al cas 2;
    break;
    ...

  default :
    bloc d'instruccions per defecte;
}
```



Si els blocs d'instruccions no són massa llargs, es pot escriure cada cas en una línia. A l'exemple següent ho hem fet així. Cal recordar que JavaScript interpreta el punt i coma (**;**) com a salts de línia. Atenció, també, amb els dos punts (**:**) obligatoris que porta cada cas.

L'exemple proposa un senzill joc d'atzar. Es tracta d'aconseguir quatre quadres del mateix color. Dins d'un bucle es crea, a cada volta, un número aleatori entre 0 i 3. Segons quin sigui, es pintarà una cel·la d'un color o un altre. Finalment, posem un enllaç per repetir la jugada.

JavaScript té l'objecte **Math** per treballar amb números. Un dels seus mètodes, **random()**, proporciona un número aleatori entre 0 i 1, sempre diferent. Si el multipliquem per 4, tenim un número decimal entre 0 i 4. Com que només ens interessa la part sencera, ho "passem" pel mètode **floor()**, que descarta els decimals. El resultat l'assignem a la variable **rnd**:

```
var rnd = Math.floor( Math.random() * 4 );
```

rnd *serà la part* ***sencera*** *d'un número* ***aleatori*** *multiplicat per quatre.*

A continuació posem el selector. Als casos 0, 1 i 2, assignem a la variable **clr** els codis de color roig, groc i verd, respectivament, i *trenquem* l'estructura. Per defecte, assignem el blau.

Un cop feta la selecció (una per cada volta del bucle) tenim la variable "rnd" amb un número i la variable "clr" amb un codi de color. Dibuixem una cel·la d'aquest color i escrivim el número.

Finalment, tanquem la fila i en fem una altra, de quatre cel·les d'amplada, amb un enllaç que torni a carregar la pàgina. D'això se n'encarrega el mètode **reload()** de l'objecte **location**. Per acabar-ho, tanquem cel·la, fila i taula.

```
<table border="0" cellspacing="4" cellpadding="30"><tr>
<script language="javascript">
for ( var i=0 ; i<4 ; i++ ) {
  var rnd = Math.floor( Math.random() * 4 );
  switch ( rnd ) {
    case 0: clr = 'FF6666'; break;
    case 1: clr = 'DDCC00'; break;
    case 2: clr = '00CC00'; break;
    default: clr = '0099FF';
  }
  document.writeln( '<td bgcolor = "#' + clr + '">' + rnd + '</td>' );
}
document.writeln( '</tr><tr><td colspan="4" align="center">' );
document.writeln( '<a href="javascript:location.reload()">Tornar-ho a provar</a>' );
document.writeln( '</td></tr>' );
</script>
</table>
```

5.- Noms de variables

Les normes són ben simples: una variable (o una funció, les veurem al mòdul següent) ha de tenir un nom, format per lletres i números - sense espais en blanc -, que no pugui entrar en conflicte amb qualsevol altre.

Tot i que no és norma, els noms de variables i funcions han de ser curts i significatius. Una tècnica, força efectiva, és l'anomenada "mescla de paraules" (wordmixing), que es basa en encadenar bocins de paraules abreujades. Hi ha qui, a més, comença amb una inicial minúscula que identifica el tipus de variable: a (Arrays), o (objectes), n (números), c (cadenes de text) , b (booleans, lògics), ... : aLnk, oWin, nInd, cTxt, bLog, ...

En general, els noms més curs es reserven per a variables amb un àmbit petit. Per exemple, s'utilitzen noms d'una lletra (i, j, k) per als índex dels bucles - que es declaren dins del bucle - i, en canvi, noms més significatius per a variables que afecten tota la pàgina.

En aquest mòdul hem vist algunes declaracions bàsiques del JavaScript. Tenen en comú que totes controlen el flux del programa.

Els bucles (**for** i **while**), amb l'ajut dels **Arrays** i **objectes**, permeten simplificar les estructures repetitives i estalviar temps.

Amb els condicionals i selectors podem dirigir el flux en un o altre sentit, de tres maneres: preguntant l'usuari (**prompt** i **confirm**), de forma aleatòria (**random**) i a partir de les dades del programa. Tot això amb eleccions binàries (**if...else**) o multiopció (**switch**).

També hem introduït alguns conceptes força útils: declaració, assignació, operacions i comparació de variables i valors; constructors (**new**), propietats (**lenght**, **href**, ...) i mètodes (**parseInt**, **indexOf**, **reload**,...) dels objectes; trucs, receptes i, fins i tot, un petit joc.

És l'hora de posar en pràctica els aprenentatges.

Exercici

Es tracta de fer:

1.- Els enllaços dels mòduls a la pàgina principal, utilitzant JavaScript (tindrem el codi).
2.- Els altres enllaços de la pàgina principal.
3.- La pàgina "index.htm" del mòdul 2, amb enllaços a quatre (o més) pràctiques.
4.- Les pràctiques enllaçades, evidentment.

En primer lloc, farem el menú d'enllaços als mòduls de la pàgina d'entrada, en JavaScript. Utilitzem la tècnica que hem vist al segon exemple de la pràctica 2.

Obrim la pàgina amb el Dreamweaver, ens situem a la "vista de código" i busquem la taula dels menús, (a l'esquerra si hem seguit el model), l'esborrem i, en el seu lloc, hi posem això:

```
<table width="168" border="0" cellspacing="1" cellpadding="2">
<script language="javascript">
<!--
var menu = new Array(
"m1/index.htm", "Programar la web",
"m2/index.htm", "Estructures JavaScript",
"m3/index.htm", "L'usuari en acció",
"m4/index.htm", "Atenció a la diversitat",
"m5/index.htm", "El temps al meu costat",
"m6/index.htm", "L'aspecte visual",
"m7/index.htm", "Interacció i comunicació",
"m8/index.htm", "Calaix desastre");
for( var i=0; i<menu.length; i=i+2 ){
  var url = menu[i];
  var nom = menu[i+1];
  document.writeln( '<tr><td class="menu"><a href=" ' + url + '"> ' ) ;
  document.writeln( '<b>' + ((i/2)+1) +'. </b>' + nom + '</a></td></tr>' ) ;
}
// -->
</script>
</table>
```

Bé, en aquest cas només es tractava de copiar i enganxar, perquè l'Array i el bucle ens els hem trobat fets. A partir d'aquest model, haurem de refer també els menús que hi ha a la dreta de la pàgina, que contenen enllaços a llocs relacionats.

Finalment, haurem de refer la pàgina "index.htm" de la carpeta "m2". Hi posarem enllaços als arxius de les pràctiques que hem guardat (quatre, com a mínim). Podem tornar a utilitzar el bucle JavaScript que hem fet a la pàgina principal, amb les variacions que calgui. També

cal recordar que tenim un full d'estils a la rel que podem enllaçar.

Pel que fa als quatre (o més) arxius enllaçats, poden ser variacions (experimentar és la millor manera d'aprendre) sobre exemples de les pràctiques. Es poden anomenar de qualsevol manera, tot i que, és clar, es recomana "m2p1_1.htm", "m2p1_2.htm", ...

Recomanacions

Utilitzem, sempre que sigui possible, les tècniques de copiar i enganxar amb el teclat: *Control+C* i *Control+V* respectivament. Així ens hi acostumarem.

En acabar aquest mòdul hem de tenir clares les estructures i la seva lògica. Si encara no hem memoritzat prou bé les instruccions i la sintaxi, no cal que ens preocupem, és normal. Durant el curs utilitzarem moltes vegades el que hem fet aquí, i ho incorporarem als nostres coneixements sense cap esforç.

Envieu un missatge a la vostra tutoria un cop l'exercici estigui penjat a Internet. Indiqueu sempre l'adreça de forma que s'hi pugui accedir directament.

L'usuari en acció

Ja hem arribat al mòdul 3, fins ara hem vist les estructures bàsiques de programació. Aquestes són comunes a gairebé tots els llenguatges de programació, tant si són per a llenguatges d'àmbit general, com si són d'entorns reduïts, com ara el navegador web.

Continuarem veient conceptes genèrics de programació, però alguns ja són específics de JavaScript i la seva interacció amb el codi HTML de les pàgines web i l'usuari.

Tots són aspectes comuns a qualsevol aplicació en JavaScript i per tant els veurem de forma regular al llarg del curs. És important assolir bé aquests conceptes, però, si us costa arribar als detalls, no us preocupeu, en la resta de pràctiques del curs s'incideix de forma indirecta amb tots aquests conceptes.

Començarem veient a la **pràctica 1**, primer contacte amb els *events* i les *funcions*. Els primers seran els elements clau per determinar quines són les accions efectuades per l'usuari i els segons per indicar què ha de fer el programa quan ha detectat un *event*. Encara que les *funcions* tenen una utilitat molt més important: l'*estructuració* del codi.

A la **pràctica 2** veurem els *formularis.* Aquests elements són els clàssics per enviar informació de l'usuari al WebMaster. En aquesta pràctica, però, analitzarem una nova funcionalitat: permetre establir una comunicació entre l'usuari i les aplicacions JavaScript, i alguna cosa més...

A la **pràctica 3** ens dedicarem a ordenar les idees. Fins ara hem estat utilitzant assíduament algunes *variables* i *operadors*, es tracta, doncs, de tenir una visió més global de tots aquests elements indispensable per qualsevol aplicació de programació.

La **pràctica 4** ens mostrarà com podem crear *finestres* filles a partir de la finestra en la que estem visualitzant el document base. Per que crear noves finestres? La raó és clara: mostrar una nova informació o una sol·licitud de dades diferenciada de la que tenim dins la seqüència establerta dins la pàgina base.

En aquest mòdul:

Pràctica 1: **Events i funcions**
Pràctica 2: **Entrada de dades**
Pràctica 3: **Variables i operadors**
Pràctica 4: **Obre la finestra**

Exercicis: **Comparacions aleatòries**

Per fer les pràctiques, obrim el Dreamweaver i ens situem a la "Vista de código", al mig de la secció <body> que, normalment, és a la línia 8. Allí hem d'inserir el codi dels exercicis. Si no tenim activada la numeració de línies, anem al menú *Ver - Opciones de vista de Código - Números de líneas* i l'activem.

Quan escrivim codi JavaScript, cal recordar que Dreamwever disposa d'una eina, el depurador, que ens facilita la detecció i el tractament d'errades. Es pot activar des del menú "Archivo - Depurar en navegador" o fent clic al botó de la barra d'estat "Vista previa/depurar en explorador".

Fins ara hem vist totes les accions realitzades amb JavaScript com a part de la secció body, de tal manera que quan es carrega la pàgina, s'executa el codi corresponent de forma gairebé instantània. En aquesta pràctica s'introdueix dos elements més del llenguatge: les **funcions** i els **events**.

Sobre els events es farà un anàlisi introductori, per poder-los utilitzar conjuntament amb les funcions, posteriorment, al mòdul 7 pràctica 1 s'analitzaran amb més detall.

Per **event** s'entén qualsevol cosa que passi en l'entorn provocada per les accions del propi usuari o bé pel canvi d'estat d'algun objecte.

Per exemple, quan l'usuari fa clic a sobre d'un botó amb el ratolí esta provocant un event, un altre exemple podria ser quan una pàgina acaba de carregar-se completament al navegador.

Quan estem interaccionant amb una pàgina, s'estan produint events constantment, ara bé, això no vol dir que a nosaltres, com a programadors de la web, ens hagin d'interessar tots i per tant no en farem cap cas a la majoria d'ells.

Però, si algun event interessa tractar-lo d'una forma específica, l'haurem de detectar mitjançant els **controladors d'events**, i posteriorment s'hagi detectat, efectuarem unes accions determinades que, generalment, estaran descrites dins una funció.

Les **funcions** ens serviran per agrupar blocs de codi de programa que realitzaran unes accions específiques. Aquests blocs de codi tindran un nom: el nom de la funció.

Les funcions han de ser d'ús general, es a dir, les hem de poder utilitzar en múltiples situacions similars. Cada situació ha de poder tenir uns valors diferents a tractar per part de la funció. Per facilitar l'afirmació anterior es disposa dels **paràmetres** que passarem a la funció.

Elements del llenguatge en aquest capítol.

Declaració **function :** Declara el bloc de codi que correspon a una funció.
Controlador d'events **on...** : Permet detectar i tractar un determinat event.
Sentència **return** : Ens retorna un valor resultant de la funció al lloc on s'ha invocat.
Objecte **Math** : Ofereix mètodes per fer tractaments matemàtics dels valors numèrics.
Mètode **round** : Arrodoneix un valor amb decimals al valor sencer més proper.
Mètode **random** : Retorna un valor numèric aleatori entre 0 i 1.

1.- Els events. Ha passat alguna cosa?

Es diu de JavaScript que és un llenguatge de programació orientat a events. Però, què és un event? Un event és qualsevol cosa que passa en el l'àmbit d'acció del programa.

Per exemple, cada cop que fem un clic amb el ratolí , passem el ratolí per sobre d'un objecte, canviem la grandària d'una finestra, es tanca una finestra, etc. s'està provocant un event. La majoria dels events que es produeixen no tenen cap interès en el funcionament del programa i són ignorats.

Però, en algunes situacions ens pot interessar que el programa faci algunes accions específiques quan es produeix un determinat event. En aquests casos, hem de detectar-lo i fer que el programa actuï en conseqüència.

La següent llista ens mostra algun dels events més habituals :

blur.- es produeix quan un element de formulari perd el focus, per exemple, quan es fa clic a un altre lloc diferent d'aquest element.

click.- es produeix quan es fa un clic amb el ratolí a sobre d'un element.
change.- es produeix quan l'usuari canvia el valor d'un camp de formulari.
focus.- es produeix quan un element rep el focus.
load.- es produeix quan es carrega una pàgina en el navegador.
mouseover.- es produeix quan l'usuari mou el cursor per sobre d'un vincle d'hipertext.
submit.- es produeix quan es fa clic a sobre del botó per a enviar les dades d'un formulari.
unload.- es produeix quan l'usuari abandona una pàgina.
etc.

Per detectar un event s'utilitza els **controladors d'events**, aquests elements de programa tenen una estructura com la següent:

onXXXX = codi d'accions a realitzar;

Les lletres XXXX indiquen el tipus d'event a detectar i el codi indicat a l'esquerra de símbol = serà l'acció a realitzar quan es produeixi aquest event.

Els controladors d'event corresponents als events indicats anteriorment són els següents.

onBlur, onClick, onChange, onFocus, onLoad, onMouseOver, onSubmit i onUnLoad

Aquests controladors s'apliquen dins les directives HTML que els poden suportar amb l'estructura indicada prèviament.

S'ha de tenir en compte que si un element de codi HTML que incorpora un controlador d'event està inclòs dins una altre element que també incorpora un altre controlador d'event, l'element inclòs heretarà també l'event de l'element que l'inclou. A aquest efecte se l'anomena **bombolleig d'events**.

Si el codi que ha d'executar un controlador d'events és molt extens, no s'acostuma a incloure a la dreta del signe = . Habitualment, se sol definir una funció que inclou tot el codi que ha d'executar l'event i solament s'assigna el nom de la funció al controlador d'events. En els apartats següents es pot veure com es treballa amb funcions i events.

El següent exemple ens mostra com són utilitzats alguns dels events indicats prèviament. L'efecte que produeix la detecció de cada event és molt simple: ens traurà un missatge d'alerta quan l'event sigui detectat. El codi a executar serà el mètode "alert" ubicat després del signe =.

Podem observar com dins la directiva <body> hi ha situats els events onLoad i onUnload.

```
<body ….. onLoad="alert('S\'ha carregat la pàgina.');"  onUnload="alert('Adéu.');">
```

Aquests events mostraran un missatge quan es carregui la pàgina i quan es descarregui.

Dins la primera fila de la taula hi ha definits dos events, l'un dins de <td> i l'altre dins de <p>, com que <p> està inclòs dins de <td> es produeix un efecte de **bombolleig d'events** al fer un clic a sobre del text "Text 1", per tant, es mostraran dos missatges d'alerta.

```
<tr … ><td … onClick="alert('Bombolleig d\'events.');">
        <p onClick="alert('Has fet un clic a sobre de Text 1.');">Text 1.</p></td></tr>
```

A la segona fila de la taula s'hi aplica el controlador d'event onMouseover, per tant, cada cop que passem el ratolí per sobre se'ns mostrarà el corresponent missatge d'alerta.

```
<p onMouseOver="alert('Has passat el  ratolí per sobre de Text 2.');">Text 2.</p>
```

Dins el formulari, s'apliquen quatre controladors d'events, el primer és **onSubmit**, que s'activarà quan fem clic a sobre el botó "Clica'm". L'event **onChange** el podrem observar cada cop que canviï el contingut del "Quadre 1" i l'event **onBlur**, s'activarà cada cop que el "Quadre 2" perdi el focus.

Els events sobre "Text 1" i "Text 2" no es podran observar amb el navegador Netscape 4.x. Aquest navegador reconeix molts menys events i efectua un tractament dels events de forma molt diferent a com ho fan els navegadors Internet Explorer i Netscape 6.x.

```
<html>
<head>
<title>Exemple</title>
</head>
<body bgcolor="#E0E8E0" onLoad="alert('S\'ha carregat la pàgina.');"
                                                    onUnload="alert('Adéu.');">
   <table border="1" align="center">
   <tr align="center"><td bgcolor="#ffffcc" onClick="alert('Bombolleig d\'events.');">
      <p onClick="alert('Has fet un clic a sobre de Text 1.');">Text 1.</p></td></tr>
   <tr align="center"><td bgcolor="#66ffcc"><p onMouseOver="alert('Has passat el
                                       ratolí per sobre de Text 2.');">Text 2.</p></td></tr>
   </table>
   <center>
   <form onSubmit="alert('Ho sento, aquest formulari no pot enviar res.');
                                                                return false;">
   Quadre 1: <input type="text" onChange="alert('Ha canviat el contingut del
                                                            Quadre 1.');"><br>
   Quadre 2: <input type="text" onBlur="alert('El Quadre 2 ha perdut el focus.');"><br>
   <input type="submit" value="Clica'm">
   </form>
   </center>
</body>
</html>
```

2.- Què són les funcions?

Una funció és un bloc de codi al que li donem un nom específic: el nom de la funció. Aquest bloc de codi realitzarà unes accions determinades i el podrem invocar tantes vegades com desitgem, des de qualsevol altre lloc de la plana web, escrivint el seu nom.

El nom de la funció sempre anirà seguit de dos parèntesi **( )**. Dins aquest parèntesi hi podrà haver, o no, els paràmetres separats amb comes **,** . Les accions que ha de realitzar la invocació de la funció han d'estar situades entre claus **{ }**.

Els paràmetres són valors que es passen a la funció, de tal manera que, el comportament de la funció pot ser modificat segons quins siguin els paràmetres que se li han passat. En una funció hi pot haver molts paràmetres dins el parèntesi o no haver-ni cap. En cas de no haver-hi cap paràmetre no es pot obviar la utilització dels dos parèntesi **( )** al final del nom de la funció.

La sintaxi genèrica d'una funció és la següent:

```
nom_funcio( paràmetre1, paràmetre2, paràmetre3, ..... )
{
     ......... Accions a realitzar per la funció ..............
     .........Finalitzem amb return únicament si s'ha de retornar algun valor.........
}
```

En qualsevol altre lloc del codi es pot invocar a la funció indicant el seu nom i els paràmetres entre parèntesi. Si la funció ens ha de retornar algun resultat el podem assignar a una variable.

Invocació sense que retorni cap valor (només realitzarà accions):

```
nom_funcio(valor1, valor2, .....);
```

Invocació retornant un valor resultant (també pot realitzar accions, a més a més):

```
variable = nom_funcio(valor1, valor2, .....);
```

S'ha de tenir en compte que els paràmetres són opcionals i que si no n'hi ha cap, els parèntesi s'han de mantenir.

Habitualment les funcions se solen definir dins el bloc *<head>* del codi HTML, encara que no hi ha cap inconvenient que estiguin definides dins el bloc *<body>*

En el següent exemple podem veure com invoquem a una funció i li passem uns paràmetres. La funció de nom **escriu** s'encarrega d'escriure a la finestra activa indicant el valors dels paràmetres que li hem passat.

La invocació de la funció es provoca quan es detecta l'event **Clic** sobre un botó, per tant en aquest exemple, també podem observar una utilització molt habitual dels events que invoquen a una funció, en lloc, d'executar el codi directament tal com s'ha fet a l'apartat sobre events.

```
<html>
<head>
<title>Exemple</title>
<script language="JavaScript">
  function escriu(parametre1, parametre2, parametre3)
  {
    document.writeln("Acabes d'invocar a la funció <b>escriu</b> i li has passat
                            els següents paràmetres:");
    document.writeln("<br>El primer paràmetre és: <font color='#FFOOOO'>"
                            + parametre1+ "</font>");
    document.writeln("<br>El segon paràmetre és: <font color='#FFOOOO'>"
                            + parametre2 + "</font>");
    document.writeln("<br>El tercer paràmetre és: <font color='#FFOOOO'>"
                            + parametre3 + "</font><p>");
  }
</script>
</head>

<body bgcolor="#E0E8E0">
<form name="form1">
<table border="0" width="370" align="center">
  <tr><td align="right">Primer paràmetre:</td><td align="center">
                            <input type="text" name="param1" value="A"></td></tr>
  <tr><td align="right">Segon paràmetre:</td><td align="center">
                            <input type="text" name="param2" value="B"></td></tr>
  <tr><td align="right">Tercer paràmetre:</td><td align="center">
                            <input type="text" name="param3" value="C"></td></tr>
  <tr><td colspan="2" align="center"><input type="button" value="Prem aquí"
             onClick="escriu(param1.value, param2.value, param3.value)"></td></tr>
</table>
</form>
</body>
</html>
```

3.- Les funcions ens poden retornar resultats.

En l'exemple anterior, hem invocat una funció passant-li uns paràmetres i ella ha fet les accions que tenia programades i que consistien en mostrar el valors dels tres paràmetres passats.
També podem fer que la funció ens retorni un resultat al lloc des d'on s'ha invocat.

El següent exemple mostra com mitjançant la sentència **return** podem retornar un valor resultant al lloc que ha invocat a la funció. La següent línia dins la funció s'encarrega de retornar el valor que s'indica a continuació de *return*.

```
return parametre1 + " " + parametre2;
```

De fet, el codi de la funció és molt simple. Retorna la concatenació (**+**)de les cadenes de text passades com a paràmetres amb un espai en blanc enmig.

Al mateix temps, des del formulari **form1** situat dins del bloc *<body>*, en el mateix lloc on s'ha invocat la funció, assignem el valor retornat a la propietat **value** del quadre de text de nom **resul**.

```
resul.value=escriu(param1.value, param2.value)
```

És a dir, en fer el clic sobre el botó, es provoca un event que és detectat pel controlador d'events **onClick**. Aquest controlador d'events invoca la funció **escriu()** i li passa com a paràmetres el contingut de **param1.value** i **param2.value**. Un cop processada la funció, aquesta retorna un resultat i s'assigna a **resul.value**.

```
<html>
<head>
<title>Exemple</title>
<script language="JavaScript">
   function escriu(parametre1, parametre2)
  {
     return parametre1 + " " + parametre2;
  }
</script>
</head>

<body bgcolor="#E0E8E0">
<form name="form1">
<table border="0" width="370" align="center">
    <tr><td align="right">Primer paràmetre:</td><td align="center">
                          <input type="text" name="param1" value="A"></td></tr>
    <tr><td align="right">Segon paràmetre:</td><td align="center">
                          <input type="text" name="param2" value="B"></td></tr>
    <tr><td align="right">Valor retornat:</td><td align="center">
                          <input type="text" name="resul" value=""></td></tr>
    <tr><td colspan="2" align="center"><input type="button" value="Prem aquí"
                    onClick="resul.value=escriu(param1.value, param2.value)"></td></tr>
</table>
</form>
</body>
</html>
```

4.- Evitem les repeticions.

Les funcions ens han de servir per reduir i estructurar el codi.

Referent a l'estructuració del codi del programa, en veurem un petit exemple descriptiu al mòdul 8 pràctica 3. Aquí, en canvi, veurem un exemple en què utilitzarem les funcions per reduir la grandària del codi.

Suposem que hem d'escriure un tros de codi que s'ha de repetir moltes vegades. Si aquest codi és extens, provocarà que el programa sigui massa llarg de forma innecessària.

La millor solució per evitar això és incloure aquest codi dins una funció i, a tots els llocs on s'hagi de repetir, invocarem la funció simplement escrivint el seu nom. Si a més, desitgem que aquest codi faci les accions de forma diferent en funció d'unes variables, podem posar a la funció els paràmetres apropiats.

El següent exemple pretén il·lustrar aquest concepte. Es pretén crear una taula on cadascuna de les caselles tingui un color diferent de forma aleatòria. Mitjançant codi, creem la taula de forma dinàmica i apliquem a cada cel·la el paràmetre "bgcolor".

El valor del color assignat a bgcolor, el determinem de forma aleatòria a partir d'una llista de colors ubicada dins un Array. Per fer la selecció aleatòria utilitzem els mètodes **random()** i **round()** de l'objecte **Math**. El codi per fer la selecció està ubicat dins la funció **color( )** que retorna el nom del color corresponent. Aquesta funció no té paràmetres.

Posteriorment, dins la taula s'invoca la funció **color()** cada cop que es desitja obtenir un color aleatori d'entre els de la llista.

Es pot apreciar que l'ús de la funció redueix bastant l'extensió del codi, ja que en cas contrari, el codi de la funció **color()** s'hauria de repetir tantes vegades com cel·les tenim, en canvi, ara solament ha calgut repetir el nom de la funció.

```
<html>
<head>
<script language="JavaScript">
    function color()
    {
        var colors=new Array("black", "blue", "brown", "fuchsia", "green",
                                                "magenta", "orange", "red", "yellow");
        return colors[Math.round(9 * Math.random() - 0.5)];
    }
</script>
</head>
```

```
<body>
<script language="JavaScript">
   document.writeln('<table border="1" width="350" align="center" >');
   document.writeln('<tr><td bgcolor="' + color() + '"> </td><td bgcolor="'
        + color() + '"> </td><td bgcolor="' + color() + '"> </td><td
        bgcolor="' + color() + '"> </td></tr>');
   document.writeln('<tr><td bgcolor="' + color() + '"> </td><td bgcolor="'
        + color() + '"> </td><td bgcolor="' + color() + '"> </td><td
        bgcolor="' + color() + '"> </td></tr>');
   document.writeln('<tr><td bgcolor="' + color() + '"> </td><td bgcolor="'
        + color() + '"> </td><td bgcolor="' + color() + '"> </td><td
        bgcolor="' + color() + '"> </td></tr>');
   document.writeln('<tr><td bgcolor="' + color() + '"> </td><td bgcolor="'
        + color() + '"> </td><td bgcolor="' + color() + '"> </td><td
        bgcolor="' + color() + '"> </td></tr>');
   document.writeln('</table>');
</script>
</body>
</html>
```

Els formularis són una forma efectiva d'establir comunicació amb els visitants de les nostres pàgines. L'altra opció, l'adreça de correu, passa sovint inadvertida. Un formulari, en canvi, convida a escriure.

JavaScript amplia les possibilitats dels formularis. Si només utilitzem HTML, els podem enviar per correu i poca cosa més. Un cop enviats, aniran a parar a una bústia o bé els rebrà un programa CGI o un script de servidor, si en tenim algun a la nostra disposició. Aquest utilitzarà les dades per a feines diverses. En tot cas, aquesta mena de programes són una ajuda externa que no sempre tindrem o podrem controlar.

Ja hem vist recursos alguns JavaScipt que faciliten la interacció: els quadres de diàleg (alert, prompt i confirm). Els formularis ens proporcionen, a més:

- Camps de text: corresponen als tipus HTML **text, textarea, password i hidden**.
- Camps lògics (marcat o desmarcat): controls **checkbox i radio**.
- Capses de selecció: **select i option**.
- Botons per enviar, esborrar i el que vulguem: **submit, reset i button**.
- Tria d'arxius: tenim el camp **file**, si utilitzem un navegador actualitzat.

El navegador maneja un Array - document.forms - que conté tots els formularis de la pàgina Cada element de l' Array, al seu torn, té tots els camps i les seves propietats. Des de JavaScript podem manipular aquesta estructura. Les possibilitats són enormes.

Conceptes JavaScript en aquest capítol

Objecte **document.forms :** matriu que conté els formularis d'una pàgina.
Operador **+= :** afegeix el valor de la dreta a la variable de l'esquerra.
Mètode **focus() :** enfoca, situa el cursor sobre un element.
Propietat **value :** el valor contingut en un camp de text o file.
Propietat **selectedIndex** : l'element seleccionat d'un control de tipus select - option.
Propietat **options[n].text :** el text que acompanya cada opció d'un control tipus select.

1.- Escriure a la pàgina des d'un formulari

Per començar, cal observar que un formularis té, normalment, una acció associada (action) i un botó que l'executa (submit). Això és massa genèric i, de moment, ens serveix de poc.

Per tal de manejar amb més flexibilitat les dades dels camps, haurem de manejar events. Com que aquests són particulars i diferents per a cada camp, tindrem control totals sobre el que passa a cada un, a més del control general sobre el formulari.

La manera més efectiva, però no única, de tractar el contingut del formulari, és interceptar l'event **onClick** d'un simple botó (tipus **button**). L'utilitzarem en lloc del típic submit. En aquest cas, com que l'exemple és molt senzill, no utilitzarem cap funció. L'únic que volem és escriure el contingut d'un camp de text.

Aquesta no és una bona manera de plantejar les coses. La utilitzem aquí perquè resulta més fàcil d'entendre. Normalment, farem una funció i passarem el formulari com a paràmetre.

En aquest cas, però, ho fem directament: document.write(nom.value), és a dir, *escriu al document el valor del camp "nom"*. Hi afegim un salt de línia, **\n**, de manera que, si volem escriure un altre text sense tornar a carregar la pàgina, ho faci sota l'anterior.

La forma correcta de referir-nos al text d'entrada és: document.forms[0].nom.value, tot i que, en aquest cas, no cal. L'índex 0 indica que es tracta del primer formulari de la pàgina. Una manera més elegant (i segura, sempre podem inserir-ne un altre abans del que hi ha) seria posar-li un nom: <form name="form1">. Després podem referenciar el valor d'un camp utilitzant la sintaxi document.form1.nom.value. Ho veurem més endavant.

```
<form>
  <input type="text" name="nom" size="32" maxlength="60">
  <input type="button" name="enviar" value="Mostrar"
      onClick="document.write(nom.value)+'\n')">
</form>
```

2.- Escriure en un altre camp i netejar l'entrada

L'exemple anterior no neteja el camp d'entrada. Si es vol escriure un altre text, s'ha d'esborrar a mà. A més, escriu directament a la pàgina, i això poques vegades serà possible.

Com que tenim a l'abast tota l'estructura del formulari, ho aprofitarem per enviar el que escrivim a un altre camp. Podem assignar el valor d'un camp a un altre, o el podem afegir.

En aquest exemple utilitzarem un camp multilínia (textarea) per anar afegint el que escrivim en un camp de text simple. Com abans, utilitzem l'event **onClic** per indicar què s'ha de fer:

```
result.value += nom.value + '\n' ; nom value = ' '
```

Hi ha dues instruccions, separades pel (;). En primer lloc afegim (amb l'operador **+=**) el valor del camp **nom** al del camp **result**, més el signe de salt de línia. Després assignem una cadena buida al valor de **nom**. Això farà que s'esborri.

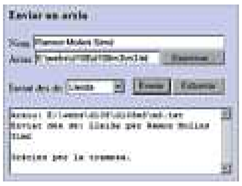
```
<form>
  <input type="text" name="nom" size="32" maxlength="60">
  <input type="button" name="enviar" value="Mostrar"
      onClick="result.value += nom.value +'\n' ; nom.value=' '"><br>
  <textarea name="result" rows="8" cols="40"></textarea>
</form>
```

3.- Utilitzar una funció i validar els camps

Tractar l'event onClick amb una funció dóna més flexibilitat i possibilitats. D'una banda, podem posar-hi més instruccions i es llegiran millor. De l'altra, podem usar la funció, que s'escriu només una vegada, des de diversos llocs de la pàgina o, si l'escrivim en un arxiu extern, accedir-hi des de diverses pàgines.

En el disseny d'una funció d'aquesta mena, hem de considerar l'ús que en farem. Si només actua sobre un formulari, no cal que tingui massa paràmetres; en canvi, si ha de controlar diversos formularis o pàgines, l'haurem de parametritzar, per tal que sigui més flexible.

Farem un formulari que demani el nom de l'usuari i li permeti triar un arxiu per enviar, a més d'especificar des d'on l'envia. De fet, no s'envia enlloc - només és un exemple - només és un quadre de text, per tal de veure'n el resultat. Hi posem dos botons, un per executar la funció i un altre per esborrar les dades. El primer passarà el control a la funció processa().



La funció rep el nom del formulari i comprova que cap dels camps estigui buit. S'encarrega de fer la comprovació una estructura condicional **if()** que ve a dir, en el cas del nom:

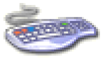
Si el valor del camp "nom" del formulari és buit, alerta que no s'ha escrit el nom, enfoca aquest camp i retorna el control al formulari.

I, així, per a cada camp que vulguem validar. "Enfocar" (**focus**) un element és centrar-hi l'atenció o, en el cas dels formularis, situar-hi el cursor. Així facilitarem que els usuaris puguin escriure o triar directament, sense haver-hi de fer clic abans.

Al camp select, que tria el lloc des d'on s'envia, farem la comprovació utilitzant la propietat **selectedIndex**, numèrica, que indica quina és l'opció seleccionada. La primera opció és la zero que, intencionadament, hem deixat buida i seleccionada per defecte. Si l'opció escollida és aquesta, aleshores no s'ha especificat el lloc, i hem d'alertar.

Quan volem saber què s'ha triat en un componen "select", com aquest, utilitzem a propietat "text", que és el text que mostra cada opció: formulari.camp.options[*número*].text.

Finalment, si tots els camps són plens, s'escriu al quadre "result" un text basat en els valors que ha proporcionat l'usuari. Utilitzem '\n' per indicar un salt de línia dins del camp. També hem d'utilitzar la fórmula de l'antibarra a les alertes, per indicar que posem un apòstrof.



```
<script language="javascript">
function processa( f1 ){
  if (f1.nom.value =='') {
    alert('No has escrit el teu nom')
    f1.nom.focus()
    return false
  }
  if (f1.arxiu.value =='') {
    alert('Tria l\'arxiu que vols enviar')
    f1.arxiu.focus()
    return false
  }
  if (f1.lloc.selectedIndex ==0) {
    alert('Has d\'especificar un lloc')
    f1.lloc.focus()
    return false
  }
  f1.result.value = 'Arxiu: ' + f1.arxiu.value + '\n'
  f1.result.value += 'Enviat des de ' + f1.lloc.options[f1.lloc.selectedIndex].text
  f1.result.value += ' per ' + f1.nom.value + '\n'
  f1.result.value += '\nGràcies per la tramesa.'
}
</script>
</head>

<body ... >
<h3>Enviar un arxiu</h3>
<form name="form1">
  Nom: <input type="text" name="nom" size="42" maxlength="60"><br>
  Arxiu: <input type="file" name="arxiu" size="28"><br><br>
  Enviat des de: <select name="lloc">
  <option selected></option>
  <option>Barcelona</option>
  <option>Girona</option>
  <option>Lleida</option>
  <option>Tarragona</option>
  </select>  
  <input type="button" name="enviar" value="Enviar" onClick="processa(form1)">
  <input type="reset" name="reset" value="Esborrar"><br><br>
  <textarea name="result" rows="6" cols="40"></textarea>
</form>
...
```

4.- FerÍndex: un exemple d'ús dels formularis

L'eina **FerÍndex** crea una pàgina d'enllaços en una carpeta amb arxius numerats, del tipus pag001.htm, pag002.htm, ... Utilitza un formulari per establir algunes característiques de l'índex i veure'n el resultat.

Tot i que, en aquest moment del curs, resultaria complicat entendre la codificació, sí que ens podem fixar en el funcionament. Es tria una pàgina model, es tria el número de pàgines que es vol analitzar, les columnes que ha de tenir el llistat, el tipus de codificació (HTML o JavaScript) i es prem un botó que engegarà la funció (en aquest cas, les funcions) per crear el codi. Tot això a la part superior de la pantalla.

El formulari té una segona part (a sota), destinada a l'edició d'aquest codi i la creació de la pàgina en una finestra nova. Això ho veurem més endavant.

Podem engegar FerÍndex en qualsevol moment des del menú d'Eines. Cal tenir una carpeta amb pàgines numerades i títols diferents. Si no en tenim cap a mà, podem provar-ho sobre la carpeta d'un mòdul del curs o, millor encara, sobre una còpia d'aquesta carpeta.

Ja s'han fet unes quantes pràctiques en què s'han utilitzat les variables i els operadors. Quan ha estat necessari utilitzar-ne algun s'ha fet la corresponent descripció local d'aquest element.

Tot i això, és necessari veure una visió més global d'aquests elements, és a dir tots ells en conjunt. Les variables, tipus de dades i operadors de JavaScript són molt similars a les que s'utilitzen en C/C++ i Java. Però, tenim la sort que es simplifica molt el seu ús i són més flexibles, ja que no és necessari declarar expressament el seu tipus. Es diu per això que JavaScript és un llenguatge amb variables de 'tipus lliure'.

En aquesta pràctica es pretén donar una descripció àmplia d'aquests elements i a la vegada veure'n uns altres de nous, classificats en els diferents tipus que existeixen. És una pràctica eminentment teòrica i la utilització del seu contingut es practica al llarg de tot el curs.

Conceptes JavaScript en aquest capítol

Tipus de dades : Classificació de les dades segons la seva naturalesa.
Sentència **var :** Declara una variable.
Variables : Elements de programació encarregats d'encabir les dades.
Operadors : Elements de programació que realitzen operacions amb les dades o constants.
Caràcters d'escape : Caràcters especials que es poden afegir dins les cadenes.
Mètode **parseInt() :** Retorna el valor numèric d'una cadena amb dígits en format enter.
Mètode **parseFloat() :** Retorna el valor numèric d'una cadena amb dígits en format coma flotant.

1.- Variables.

Les variables són uns elements de programació encarregades de contenir les dades que s'utilitzen en el programa. El seu valor pot canviar en funció de com evolucioni la seqüència del programa.

Per declarar una variable s'utilitza la paraula reservada **var**. Aquesta indica que el nom que va a continuació és una variable i li reserva l'espai corresponent a la memòria. La declaració d'una variable es pot fer al principi del programa, o bé, entremig del programa en el moment que es produeix la seva primera utilització.

A continuació de la paraula *var*, s'indica el nom que tindrà la variable. Aquest nom ha de ser únic i representatiu del valor que contindrà la variable. També se li pot afegir una assignació inicial.

```
var camí;
var setmana = 0;
```

Els noms de les variables han de tenir com a primer caràcter una lletra o bé el guió baix (_), el nom pot ser qualsevol, però, no ha de coincidir amb cap nom igual als de les paraules reservades del llenguatge i no pot contenir espais o caràcters especials.

JavaScript diferencia entre majúscules i minúscules. Per exemple, no és la mateixa variable "mostraValor" que "mostravalor".

Les següents declaracions de variable són incorrectes, ja que el nom no compleix els requisits indicats anteriorment:

var 1dia;	Incorrecte, ja que el primer caràcter és numèric.
var temps passat;	Incorrecte, ja que s'ha inclòs un espai en blanc.
var temps-passat;	Incorrecte, ja que s'ha inclòs un caràcter especial.

En una mateixa línia es pot declarar més d'una variable:

var dia1, temps_passat, hora, setmana=0;

A les variables se'ls assignen les dades. Les dades es classifiquen segons quin sigui el seu tipus. Els diferents **tipus de dades** són:

enters, coma flotant, hexadecimals i octals, booleans, nuls i cadena

Quan es declara una variable, si no se li ha assignat cap valor inicial, contindrà el valor **null**. Posteriorment., quan se li assigna un valor, la variable passarà a tenir assignat un determinat tipus de dades. El tipus dependrà de quin sigui el valor assignat inicialment. A continuació tenim una breu descripció de cada tipus de dades:

"Enter", aquest tipus correspon a dades numèriques formades únicament per valors enters. Exemple: 25, -234, 7000239, -234000123, etc.

"Coma flotant", aquest tipus correspon a dades numèriques amb decimals. En aquest tipus diferenciem la part sencera de la decimal mitjançant la utilització d'un punt separador. També en forma part els valors exponencials mitjançant una "e" o "E" per indicar l'exponent. Per exemple, els següents valors són números de coma flotant. 25.0, 0.23452, -1234.9876, -18374592.938457, 3e5, -8E-12.

"Hexadecimals i Octals". Si es desitja representar un número en notació hexadecimal ho farem indicant a davant del número els caràcters 0x. Per exemple. 0x3832, 0xFA74, etc. En el cas dels números en notació octal, s'ha de posar un zero al davant. Per exemple: 0374, 01264, 0227, 0202.

Recordem que els dígits del sistema de numeració hexadecimal són 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E i F, i els del sistema de numeració octal són: 0, 1, 2, 3, 4, 5, 6 i 7.

"Booleà" Aquest és un tipus de dades molt usat en programació. Els dos possibles valors que pot tenir són **true** i **false**. Aquests valors són els que se solen obtenir quan s'efectuen comparacions lògiques entre dos o més elements.

"Nul" Aquest és el valor **null**, un valor que indicarà que a una variable no se li ha assignat cap valor.

"Cadena", s'entén per cadena a un conjunt de caràcters tipus lletres, espais, números i símbols tancat entre cometes dobles (") o entre cometes simples ('). Uns exemples de cadenes correctes i incorrectes poden ser els següents:

Correcte	Incorrecte
"Això és una cadena"	"Cadena incorrecta'
'Això també és una cadena'	"Cadena "incorrecta"
'Aquesta cadena es "correcta" ja que les cometes inicials són també les del final'	"Cadena ' incorrecta " '
"2354"	
"Preu = 25 €"	

Les cadenes han d'estar tancades entre el mateix tipus de cometes (simples o dobles), és a dir, les que s'utilitzen al principi són les que han de servir per tancar la cadena i en el seu interior no s'hi pot trobar les mateixes cometes, però, si que hi podem trobar les cometes de l'altre tipus.

Si dins una cadena es vol introduir algun caràcter especial com ara un salt de línia o unes cometes simples o dobles, una barra invertida, etc. es pot fer utilitzant els anomenats **caràcters d'escape** propis del llenguatge C. Entre els més habituals hi ha els següents:

\n : introdueix un salt de línia.
\' : introdueix unes cometes simples.
\" : introdueix unes cometes dobles.
**** : introdueix una barra invertida (antibarra).

Aquests elements són molt utilitzats dins les cadenes de text, permeten introduir salts de línia, també permeten introduir cometes simples i cometes dobles quan els

delimitadors de la cadena també són cometes simples o dobles, respectivament.

Els següents exemples mostren la utilització d'aquests caràcters d'escape:

La cadena …	Es veurà com …
"Aquí tenim un salt \n de línia."	Aquí tenim un salt de línia.
'Cometa simple a l\'interior.'	Cometa simple a l'interior.
"Aquesta \"cadena\" conté \". "	Aquesta "cadena" conté ".

Per acabar aquest apartat, veiem més exemples de declaració de variables, a la columna de l'esquerra hi tenim unes declaracions de variables no inicialitzades (es podran inicialitzar posteriorment, de moment contenen el valor "null"), per tant no tenen un tipus de dades definit.

En canvi en la columna de la dreta hi veiem unes variables inicialitzades: les dues primeres són tipus "cadena", la tercera és tipus "coma flotant", la quarta és tipus "enter" i l'última és tipus "booleà":

Variables no inicialitzades	Variables inicialitzades
var nom;	var nom = "Ramon";
var cadena;	var cadena = 'El preu de cost dels productes';
var preu;	var preu = 223.14;
var quantitat;	var quantitat = 12;
var esMajor;	var esMajor = true;

2.- Variables locals i globals.

Segons sigui el lloc on es declarin les variables, aquestes es poden classificar en variables *globals* o *locals*.

Les variables **globals** es declaren fora de qualsevol funció. Aquestes variables poden ser utilitzades des de qualsevol lloc de la pàgina, en scripts situats en el bloc *body* o en el bloc *head* i dins i fora de qualsevol funció. És a dir, la seva vida és global i existiran mentre la pàgina estigui carregada en el navegador.

Les variables **locals** es declaren dins d'una funció. Aquestes variables existiran solament dins d'aquesta funció i desapareixeran quan no s'estigui executant el codi de la funció.

Observeu el següent codi. És un exemple per observar el comportament de les variables *globals* i *locals*. La variable de nom **varGlobal** està definida fora de qualsevol funció, per tant, al carregar la pàgina aquesta variable quedarà declarada i el seu ús serà vàlid des de qualsevol lloc de la pàgina web, mentre aquesta estigui carregada.

En canvi, la variable de nom **varLocal**, està definida dins d'una funció. Per tant, solament es declararà quan s'executi el codi d'aquesta funció i desapareixerà quan la funció *funcioProva()* acabi la seva execució.

La funció **funcioProva()**, a més, conté un *alert()* que mostra el contingut de la variable *varLocal* i també conté dues instruccions per incrementar en una unitat a les variables *varLocal* i *varGlobal*.

El formulari que hi ha dins el bloc <*body*> conté tres botons que serveixen per mostrar el valor de les dues variables.

- El botó **1** mostra mitjançant un *alert()* el valor de la variable *varGlobal*.

- El botó **2** intenta mostrar, mitjançant el sistema del botó 1 el valor de *varLocal*, però, sempre dóna error, indicant que aquesta variable no existeix.

- El botó **3** executa la funció *funcioProva()*, per tant, es declara localment a la variable *varLocal* i es mostra el seu contingut mitjançant *alert()*, també s'incrementa a *varLocal* i *varGlobal* en una unitat.

Observeu que sempre que es prem el botó 3, el valor que es mostra per a *varLocal* sempre és 0, malgrat existir la línia ++varLocal; . El motiu és perquè aquesta variable, tot i que és incrementada, desapareix en acabar la funció *funcioProva()* i la propera vegada es fa clic sobre el botó 3, aquesta variable es torna a crear de nou i se li assigna el valor 0 i el torna a mostrar.

En canvi, si hem polsat múltiples vegades al botó 3, i posteriorment polsem sobre el botó 1, observarem que el valor de la variable *varGlobal* si que s'ha incrementat, ja que aquesta variable es accessible des de qualsevol lloc de la pàgina web i l'increment provocat per la línia ++varGlobal es manté encara que hagi acabat la funció *funcioProva()*.

```
<html>
<head>
<script language="JavaScript">

    var varGlobal=0;

    function funcioProva() {
        var varLocal=0;
        alert(varLocal);
        ++varGlobal;
        ++varLocal;
    }

</script>
</head>
<body>
<form>
  <input type="button" value="1- Mostra varGlobal" onClick="alert(varGlobal);"><p>
  <input type="button" value="2- Mostra varLocal, (error)"
                                                  onClick="alert(varLocal);"><p>
  <input type="button" value="3- Mostra varLocal, (funcioProva)"
                                                  onClick="funcioProva();">
</form>
</body>
</html>
```

3.- Operadors.

Els operadors són els elements de codi que ens serviran per fer operacions amb les variables i altres valors constants. En JavaScript s'utilitzen bàsicament els mateixos operadors que en C/C++.

Els operadors es classifiquen en: Operadors d'assignació, operadors aritmètics, operadors lògics, operadors de comparació, operadors condicionals i operadors de cadena (existeixen altres tipus, però no els tractarem ja que són d'ús molt específic).

Les taules que vénen a continuació classifiquen els diferents operadors amb una descripció i un exemple d'ús.

Operadors d'assignació:

Operador	Descripció	Exemple d'ús
=	Assigna el valor de l'operand de la dreta a l'operand de l'esquerra.	t = r ;
+ =	Assigna la suma dels operands de la dreta i l'esquerra a l'operand de la dreta.	t += r ; (t = t + r)
- =	Assigna la resta dels operands de la dreta i l'esquerra a l'operand de la dreta.	t -= r ; (t = t - r)
*** =**	Assigna el producte dels operands de la dreta i l'esquerra a l'operand de la dreta.	t *= r; (t = t * r)
/ =	Assigna el quocient dels operands de la dreta i l'esquerra a l'operand de la dreta.	t /= r; (t = t / r)
% =	Assigna el residu del quocient dels operands de la dreta i l'esquerra a l'operand de la dreta.	t %= r ; (t = residu(t/r))

Operadors aritmètics:

Operador	Descripció	Exemple d'ús
Binaris:		
+	Suma el valor de dos operands.	a + b;
-	Resta el valors de dos operands.	a - b;
*****	Multiplica el valor de dos operands.	a * b ;
/	Divideix els valors de dos operands.	a / b ;
%	Calcula la resta de la divisió de dos operands.	a % b;
Monaris:		
–	Canvia el signe de l'operand de la seva dreta.	- preu ;
++	Incrementa en un unitat l'operand de la seva dreta o esquerra.	++a ; a++;
– –	Disminueix en una unitat l'operand de la seva dreta o esquerra.	--a; a--;

Operadors lògics:

<table>
<tr><th>Operador</th><th>Descripció</th><th>Exemple d'ús</th></tr>
<tr><td>&&</td><td>Operador binari. Fa l'operació "i lògica", retorna true si els dos operands són true.</td><td>a && b;</td></tr>
<tr><td>||</td><td>Operador binari. Fa l'operació "o lògica", retorna true si un dels dos operands és true.</td><td>a | | b;</td></tr>
<tr><td>!</td><td>Operador monari. Fa l'operació "negació lògica", retorna true si l'operador és false.</td><td>! a;
!esMenor;</td></tr>
</table>

Operadors de comparació:

Operador	Descripció	Exemple d'ús
= =	Retorna true si els dos operadors són iguals.	a = = b;
! =	Retorna true si els dos operadors són diferents.	a ! =b;
>	Retorna true si l'operador de l'esquerra és més gran que el de la dreta.	a > b;
<	Retorna true si l'operador de l'esquerra és més petit que el de la dreta.	a < b ;
> =	Retorna true si l'operador de l'esquerra és més gran o igual que el de la dreta.	a > = b ;
< =	Retorna true si l'operador de l'esquerra és més petit o igual que el de la dreta.	a < = b;

Operador condicional (? :) :

Aquest és un operador ternari, la seva sintaxi és la següent:

(condició) ? valor1 : valor2

Si el contingut del parèntesi (la condició) és true, s'avaluarà el valor1, de cas contrari, si la condició és false, s'avaluarà el valor2. El següent exemple ens mostra una possible utilització.

a = (3>2) ? 10 : 2 ;

En aquesta assignació, la variable **a** contindrà el valor 10. Per determinar aquest valor, inicialment, s'avalua la condició (3 > 2), com que és certa, es retorna el valor 10 per fer l'assignació. Si la condició hagués estat falsa, s'hauria retornat el valor 2.

Operador de concatenació de cadenes:

L'operador binari **+**, és un operador de cadena que ens permet concatenar dues cadenes alfanumèriques. La seva aplicació ens retorna una nova cadena que serà la concatenació de les altres dues sobre les que opera.

Exemple:

```
var cadena1 = "Hola, ";
var cadena2 = "que tal";
var cadena3 = cadena1 + cadena2;
```

La variable "cadena3" contindrà el valor: "Hola, que tal".

4.- Barreja de tipus.

Operacions entre cadenes i valors numèrics:

En algunes ocasions és necessari concatenar cadenes amb dades que no són cadenes, per exemple, pot ser necessari concatenar un tipus cadena amb un tipus enter. En aquests casos es pot fer directament ja que JavaScript detecta aquesta situació i, automàticament, efectua la conversió del tipus numèric a cadena. En el següent exemple:

```
var perfecte = "Nota: " + 10;
```

La variable "perfecte" contindrà el valor de cadena "Nota: 10", ja que el numèric 10 s'haurà convertit automàticament, per part de JavaScript, en una cadena.

La situació contrària no és tan simple d'implementar, si es vol fer una suma algebraica d'un valor de cadena que conté dígits numèrics amb un valor numèric tipus enter, és necessari efectuar una conversió prèvia mitjançant la funció **parseInt(cadena)**. Per exemple: var numero = 3 + "10"; és incorrecte. La forma correcta seria la següent:

```
var numero = 4 + parseInt("10");       //numero valdrà 14
```

La sintaxi completa de parseInt() és: *parseInt(cadena[, base])*, on el paràmetre opcional *base* indica la base de numeració.

Pel cas dels valors tipus coma flotant tenim una altra funció equivalent: **parseFloat(cadena)**. Aquesta funció retornarà el valor numèric en coma flotant d'una cadena amb dígits numèrics en un format de coma flotant.. Per exemple:

```
var numdec = 3 + parseFloat("3.6");       //numdec valdrà 6.6
```

Window és l'objecte més alt de la jerarquia, inclou tots els altres. Cada finestra del navegador és una instància d'aquest objecte. Les seves propietats i mètodes ens informen sobre les característiques i comportaments de la finestra i, si el navegador ho permet, les podrem manipular.

Quan fem referència als elements de la pròpia finestra no utilitzem la nomenclatura completa, tot i que ho podem fer. Tant és si escrivim "location" com "window.location". Si no ho especifiquem, el navegador suposa que ens referim a la finestra activa (potser hauríem de dir "enfocada"?). En canvi, sí que és obligatori usar aquesta nomenclatura quan ens referim a una altra finestra, no la paraula "window", sinó el nom de la variable que en conté la instància:

```
novaFinestra.document.write(...
```

Més endavant, quan tractarem el DOM (Document Object Model), veurem amb més detall tota la jerarquia d'objectes JavaScript.

Conceptes JavaScript en aquest capítol

Objecte **window :** la finestra.
Declaració **with() :** estableix l'objecte base per a un conjunt d'instruccions.
Mètode **open() :** obre una nova finestra del navegador.
Mètode **close() :** tanca la finestra, el document o l'element que especifiquem.

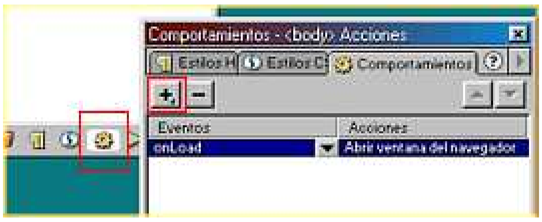
1.- Com ho fa Dreamweaver

Amb el Dreamweaver tenim el "Comportamiento" "Abrir ventana del navegador" que ens facilita la creació del codi necessari per obrir una nova finestra del navegador.

Aquesta tindrà les característiques que establim al quadre de diàleg: pàgina que obrirà, amplada, altura, barres de desplaçament, menús, eines, estat, ... si la volem maximitzar i, també, un nom. Podrem fer referència a la finestra utilitzant aquest nom, per exemple:

```
<a href="pagina1.htm" target="lamevafinestra"> ... </a>
```

Es dóna per fet que, a la finestra, hi haurà una altra pàgina i, l'únic que podem fer des de Dreamweaver o HTML és això, obrir pàgines. Si volem manipular el contingut des de la pàgina que l'ha oberta, o volem llegir informació de la finestra, necessitem JavaScript.



Obrim el Dreamweaver amb un arxiu nou.
A la barra "Lanzador" fem clic a la icona "Mostrar Comportamientos". També ho podem fer des del menú "Ventana - Comportamientos" o amb les tecles *Majús+F3*.

A la nova finestra, fem clic al botó "**+**" i triem "Abrir ventana del navegador".
Deixem en blanc "Mostrar URL", marquem algunes opcions, li posem un títol i acceptem.
Mirem ara la vista de codi:

```
<script language="javascript">
<!--
function MM_openBrWindow(theURL,winName,features) { //v2.0
   window.open(theURL,winName,features);
}
// -->
</script>
</head>
```

```
<body bgcolor="#FFFFFF" text="#000000"
onLoad="MM_openBrWindow('','win1','toolbar=yes,scrollbars=yes,resizable=yes')">
...
```

Quin és el problema? Doncs que l'única opció que ens ha deixat per referir-nos a la nova finestra és a través del HTML, com a destí (target) d'un formulari o un vincle. Per poder manejar la nova finestra des de JavaScript, necessitem una variable global. Això sí: ha deixat pistes que ens permetran crear la nostra pròpia funció.

2.- Obrir una finestra

Des d'un event (onClick, onLoad) podem obrir una nova finestra amb la declaració:

```
onLoad = "window.open( adreça , nom , característiques )"
```

En la majoria de casos, però, és millor fer que l'event es dirigeixi a una funció, així tindrem més possibilitats i control. Dreamweaver n'utilitza una que podem fer servir com a base.

Assignarem la finestra a una variable global, definida fora de la funció. En aquest exemple no caldria, però ho fem així per assegurar que ens hi podrem referir des d'altres funcions. Per fer l'exemple utilitzem l'arxiu del tercer exemple de la pràctica 2.

La funció rep com a paràmetre el nom del fitxer que s'ha d'obrir a la nova finestra. A continuació enfoca la finestra "mare" per desmarcar l'enllaç i assigna l'acció (window.open) a la variable global. Finalment tanca el document de la finestra, l'enfoca i retorna "fals".

A la secció <body> farem l'enllaç amb un petit truc: deixarem l'adreça en blanc i assignarem l'acció a levent onClick, utilitzant la forma **onClick="return(funció())"**. Com que la funció que s'executa retorna fals, el vincle buit no s'executarà.

```
<script language="javascript">
var finestra
function obreFinestra(fitxer) {
  window.focus()
  finestra = window.open( fitxer, 'win1', 'width=440,height=300,top=100,left=100' )
  finestra.document.close()
  finestra.focus()
  return false
}
</script>
</head>

<body ...>
<a href="" onClick="return( obreFinestra('m3p2_3.htm') )">Enviar arxius</a>
<br>
...
```

3.- Escriure a la finestra

En el cas anterior teníem un arxiu que s'obria en una finestra. Ara crearem una finestra sense arxiu i l'omplirem de contingut. Utilitzem, naturalment, **finestra.document.write()**. De fet, podem fer diverses accions sobre el document o altres components de la finestra:

```
finestra.element[.subelements].acció()
```

També podem consultar i, depenent del navegador, modificar, propietats, valors en camps de formularis i, fins i tot, variables contingudes en scripts. En el cas que ens ocupa, però, només es tracta d'escriure a la finestra. Agafem com a base l'exemple anterior.

La funció rep com a paràmetre un codi de color, que serà diferent segons quin sigui l'enllaç que l'ha feta obrir. A continuació s'escriu tota l'estructura d'una pàgina web senzilla. Com a color de fons utilitza el del paràmetre i, a més, l'escriu a la pàgina.

Com que hem d'utilitzar mètodes de l'objecte "document" de la finestra en diverses línies, emprem la declaració **with()**, que ens permet especificar sobre quin objecte base s'apliquen les instruccions de tot un bloc. A dintre del bloc, no cal escriure l'objecte.

```
<script language="javascript">
var finestra
function obreFinestra(clr) {
  window.focus()
  finestra = window.open( '', 'win1', 'width=300,height=150,top=200,left=200' )
  with( finestra.document ) {
    writeln( '<html>\n<head>\n<title>Colors</title>\n</head>')
    writeln( '<body bgcolor = "#' + clr + '">\n<center>')
    writeln( '<h1><br>#' + clr + '</h1>' )
    writeln( '</center>\n</body>\n</html>')
    close()
  }
  finestra.focus()
  return false
}
</script>
</head>

<body ...>
<a href="" onClick="return( obreFinestra('66CCFF') )">Finestra blava</a><br>
<a href="" onClick="return( obreFinestra('EE6666') )">Finestra roja</a><br>
<a href="" onClick="return( obreFinestra('99FF66') )">Finestra verda</a><br>
...
```

4.- Tancar la finestra

Ja hem vist com podem obrir una finestra i escriure-hi. També, com fer enllaços que passin paràmetres i com una funció els pot recollir i utilitzar. Fins ara, hem tancat les finestres fent clic en el botó (x) Tancar, o amb la combinació de tecles *Alt+F4*.

També la podem tancar des de la pròpia pàgina, o des de la que l'ha oberta. En tenim prou amb un vincle, un botó o qualsevol event que puguem interceptar.

close() és el mètode que s'utilitza per tancar una finestra. Des de la pròpia pàgina escrivim window.close(). Des de la pàgina "mare" escrivim nomFinestra.close(). En aquest cas, cal comprovar abans que la finestra existeix, si no provocarem una errada del navegador.

```
  if ( finestra ) finestra.close(); return false
```

El codi és prou curt i el podem escriure "en línia". Si ho preferim, també podem fer una funció com la d'obrir finestra. Cal recordar que l'enllaç és buit, i per això hem de retornar "false".

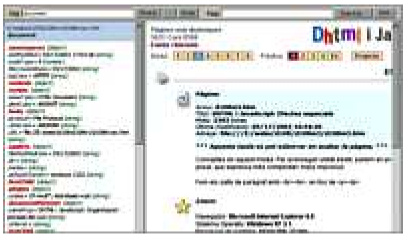
```
<script language="javascript">
var finestra
function obreFinestra() {
  window.focus()
  finestra = window.open( '', 'win1', 'width=300,heigth=150,top=200,left=200' )
  with( finestra.document ) {
    writeln( '<html>\n<head>\n<title>Nova finestra </title>\n</head>')
    writeln( '<body>\n<center>')
    writeln( '<a href="" onClick="window.close();return false">Tancar</a>' )
    writeln( '</center>\n</body>\n</html>')
    close()
  }
  finestra.focus()
  return false
}
</script>
</head>

<body ...>
<a href="" onClick="return( obreFinestra() )">Obrir finestra</a><br>
<a href="" onClick="if(finestra) finestra.close();return false">Tancar finestra</a><br>
...
```

5.- L'inspector d'objectes

Aquesta eina analitza pàgines i en mostra els objectes. Permet endinsar-se en la jerarquia i veure les propietats, és a dir, veure el mateix que "veu" el navegador.

Només hem utilitzat DHTML i JavaScript, de manera que el propi inspector pot ser objecte d'estudi, més endavant, per veure com s'ha fet.

Només pot funcionar a la mateixa màquina on és la pàgina que inspecciona, és a dir, ni el podem utilitzar des de la xarxa, ni podem analitzar pàgines a través d'una connexió.



En aquest mòdul ja hem vist uns quants elements interessants de JavaScript que permeten efectuar coses que no seria possible fer-ho solament amb HTML. Hem vist que són els events i alguns exemples d'ús, hem après i utilitzat funcions, treball amb formularis i creació de noves finestres.

Ara ho posarem en pràctica.

EXERCICIS:

Els exercicis es classifiquen en dos blocs, l'un d'implementació simple i l'altre d'implementació més complexa dels exemples de les pràctiques, on s'hi troba un apartat opcional.

Per començar modificarem la pàgina index.htm que hi ha dins la carpeta del mòdul 3 per tal que tinguem 4 enllaços amb noms: Exercici 1, Exercici 2, Exercici 3 i Exercici 4.

Cada enllaç apuntarà a una pàgina html diferent que correspondrà a quatre exercicis, un per pàgina. Heu de donar un nom a cadascuna d'aquestes pàgines, es recomana "m3p1_1.htm", "m3p2_2.htm", ... o similar.

Bloc I:

Els tres primers han de correspondre a un exercici, a elegir, dels que es troben dins les pràctiques 1, 2 i 4 respectivament amb alguna variació que faci que no siguin idèntics. Són senzills d'implementar, ja que només cal copiar-los de com estan en les pràctiques i modificar-los lleugerament.

Bloc II:

El quart exercici consta d'una pàgina que contindrà una barreja dels conceptes vistos en les pràctiques 1, 2 i 4 i part de les pràctiques del mòdul anterior.

La seva descripció és la següent:

En obrir-se la pàgina ens ha de mostrar un missatge de benvinguda i en tancar la finestra un missatge de comiat, mitjançant un *alert* (vegeu el m3p1apartat 1). Aquesta finestra ha de contenir un formulari i un text descriptiu de com s'ha fet. Opcionalment, també contindrà un enllaç per obrir una nova finestra que mostrarà números aleatoris.

El **formulari** ha de constar de tres camps de text i un botó. Els camps de text 1 i 2 han de convidar a introduir un valor de text o numèric a cadascun d'ells i al fer clic sobre el botó s'ha d'invocar una funció que escrigui un text sobre el camp de text 3 indicant si el contingut dels dos camps 1 i 2 són iguals o diferents. (guieu-vos pel m3p2 ap2 i ap3). Quan els camps 1 o 2 canviïn de valor, s'ha d'esborrar el contingut del camp 3, o sigui, rescriure'l amb ' ' (vegeu m3p1ap1).

Apartat opcional
L'**enllaç** ha d'invocar a una altra funció que sol·liciti un valor numèric *n* mitjançant un *prompt*. S'ha de validar que aquest és numèric (ha de mostrar un missatge d'error amb *alert* en cas contrari).

Si el valor numèric és correcte, ha d'obrir una finestra filla (el seu contingut no ha d'existir com a tal pàgina html) (vegeu m3p4 ap2) i, dinàmicament, s'ha d'escriure en el seu interior una taula de 10 x 10 cel·les (vegeu m2p1ap2) i cada cel·la ha de contenir un número aleatori entre 0 i *n* (vegeu m3p1ap4). També ha de contenir un link que serveixi per tancar-la (m3p4 ap4).

Els passos a seguir per fer aquest exercici no. 4 poden ser els següents:

1.- Creem la pàgina mare mitjançant l'editor del Dreamweaver o la llibreta amb la seva estructura mínima, introduïm els events onLoad i onUnload dins la directiva <body…> per crear els missatges de benvinguda i comiat.

2.- Dins el bloc <body> creem el formulari, a mà, amb els tres camps de text i el botó. Creem una nova funció, de nom *compara()*, que en invocar-la efectuï una comparació del contingut del camp 1 i camp 2 i escrigui 'són iguals' o 'són diferents' dins el camp 3. Associem aquesta funció al controlador d'event *onClick* del botó.

3.- Afegim al camp 1 i camp 2 un controlador d'event *onChange* que, quan s'activi, assigni al camp 3 una cadena buida, esborrant el seu contingut.

4.- Creeu una nova funció de nom *numsAleatoris()*, que en invocar-la sol·liciti un número mitjançant un *prompt*, comproveu amb *isNaN()* si el valor introduït és un número. Si no és un número, es mostrarà un missatge d'error amb un *alert* i acabarà la funció.

5.- Si s'ha introduït un número correctament, s'ha de crear una nova finestra filla, al crear-la, heu d'assignar una variable, de nom *finestra*, al valor retornat per *windows.open(...)*. Aquesta variable ens servirà per fer les referències posteriors a la finestra filla.

6.- Posteriorment, heu de crear dinàmicament (*for...*) la taula de 10x10 dins la finestra filla amb el mètode *write()* referenciat adequadament. Quan hagueu d'escriure el contingut de cada cel·la, l'heu d'obtenir utilitzant adequadament el mètode *random()* per obtenir el número aleatori entre 0 i *n*.

7.- Un cop creada la taula dinàmicament, heu d'introduir dins la finestra filla, un link nul que en fer-li clic tanqui la finestra. La funció ha d'acabar amb un *return false*, per evitar que s'executi el link nul que la invocarà.

8.- Heu de ficar a la pàgina un enllaç nul amb el text "Crear 100 nos. aleatoris" que enl fer-li clic, invoqui a la funció *numsAleatoris()*.

9.- Un cop funcioni adequadament tot l'exercici, introduïu a la pàgina, conjuntament amb el formulari i l'enllaç, una descripció clarificadora de com ho heu fet per al vostre tutor. Doneu-li també, els formats estètics que considereu oportuns.

Trucs, consells i receptes

La forma d'editar els exercicis és mitjançant el Dreamweaver amb la vista código o mitjançant la llibreta, visualitzant el resultat des del navegador.

L'exercici 4 és més complex. No el vulgueu fer tot de cop i provar-lo al final. Aneu per passos, seguint la seqüència indicada i al final de cada pas comproveu el correcte funcionament.

Guieu-vos amb les referències a les pràctiques indicades per a cada apartat i copieu i enganxeu el tros de codi que us faci falta i modifiqueu-lo adequadament.

Envieu un missatge a la vostra tutoria un cop l'exercici estigui penjat a Internet. Indiqueu sempre l'adreça de forma que s'hi pugui accedir directament.

Atenció a la diversitat

És clar que els humans som diferents els uns dels altres - sortosament - i que, sovint, ens costa posar-nos d'acord. Tot i que les TIC són gestionades per màquines i programes, hi hem deixat l'empremta i la diversitat és evident. A més, la tecnologia avança molt de pressa i el ritme de renovació dels usuaris, de vegades, no tant.

JavaScript va néixer amb la versió 2 de Netscape i, des d'aleshores, ha evolucionat. Al principi tenia les instruccions bàsiques per actuar sobre pàgines web, però no disposava de gaires recursos per crear efectes. Amb l'aparició dels fulls d'estils i de les capes, JavaScript s'ha adaptat per tal de poder manejar aquestes noves funcionalitats.

La versió 1.1, que acompanyava els navegadors 3.0, ja tractava imatges i Arrays. A partir de la versió 1.2 (navegadors 4.0), i coincidint amb un període de lluita pel mercat, comencen les incompatibilitats greus. Microsoft s'inventa la seva pròpia versió, JScript. Aquestes diferències es mantenen, tot i que hi ha hagut intents per suavitzar-les.

Si amb els navegadors ja hi ha diferències, amb la resta del programari i amb els equips informàtics podem trobar encara més varietat. Servidors, sistemes operatius i maquinari divers intenten veure les nostres pàgines, i tots esperen veure-les bé.

Quan dissenyem, no podem pensar que tothom té, per exemple, un monitor i una tarja gràfica com els nostres. Això ens obliga a fer diverses proves i adoptar solucions, sovint, salomòniques, perquè resulta pràcticament impossible adaptar-se a totes les possibilitats i, alhora, fer una mica de disseny.

Podem usar una gamma de colors estàndard, per exemple, la que proporciona Dreamweaver. També podem dissenyar per a 800x600 i esperar que la majoria dels usuaris tinguin aquesta resolució o superior i que, en aquest últim cas, no els molesti massa desaprofitar una part de la pantalla.

Està bé que pensem en mínims i fem el que calgui per assegurar que tothom podrà veure el que pretenem mostrar. Però, i si la pàgina tingués la capacitat de saber on és? I si, a l'hora de dissenyar, poguéssim tenir en compte els diversos entorns, i actuar en conseqüència? Doncs, alguna cosa tenim, encara que potser no tot el que voldríem.

El navegador maneja una estructura, el DOM (Document Object Model) que conté algunes informacions sobre, per exemple, el propi navegador o la pantalla, i ens proporciona uns mètodes (una interfície) per poder-les manipular ... relativament.

Veurem com es prepara una pàgina per tal que funcioni amb tots dos navegadors. A més, treballarem amb els estils i les propietats de pantalla, per aconseguir una aparença similar. Caldrà conèixer una mica millor com s'organitzen els objectes de la pàgina.

En aquest mòdul:

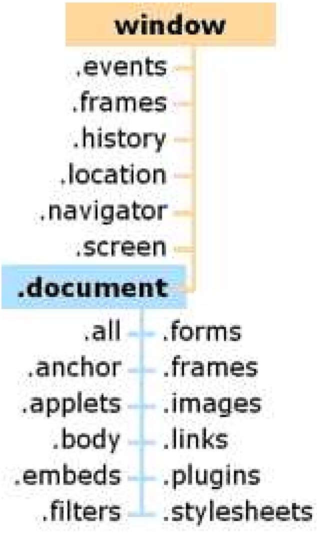
Pràctica 1: **El model d'objectes (DOM)**
Pràctica 2: **Detectar el navegador**
Pràctica 3: **Afinem els estils**
Pràctica 4: **Control de l'entorn**

Exercicis: **Informe del client**

Per fer les pràctiques, obrim el Dreamweaver i ens situem a la "Vista de código", al mig de la secció <body> que, normalment, és a la línia 8. Hi hem d'inserir el codi dels exercicis. Si no tenim activada la numeració de línies, anem al menú *Ver - Opciones de vista de Código - Números de líneas* i l'activem.

Quan escrivim codi JavaScript, cal recordar que Dreamwever disposa d'una eina, el depurador, que ens facilita la detecció i el tractament d'errades. Es pot activar des del menú "Archivo - Depurar en navegador" o fent clic al botó de la barra d'estat "Vista previa/depurar en explorador".

Atenció a la diversitat > El model d'objectes (DOM)



El Model d'Objectes del Document és l'estructura formada per tots els elements manipulables de la pàgina. Juntament amb els events proporciona la interfície que permet als scripts comunicar-se amb el navegador.

L'objecte més alt de la jerarquia és **window**. És tot el que hi ha a la finestra. Quan ens referim a la pantalla, navegador, marcs, adreça o historial, ho fem a partir de l'objecte window.

L'altre objecte important és **document**, l'amo de tots els elements de la pàgina: cos, paràgrafs, enllaços, formularis, imatges, estils, scripts, ...

Un objecte és una estructura que conté dades assignades a variables, que anomenem **propietats**, i unes funcions que manipulen aquestes variables, i que s'anomenen **mètodes**.

A partir d'aquesta estructura es crea la programació orientada a objectes, basada en l'herència de propietats i mètodes, i en el polimorfisme, que fa que les variables i funcions semblants d'objectes diferents puguin tenir el mateix nom. Java i C++ són llenguatges orientats a objectes, el JavaScript també ho és, tot i que més simplificat i flexible.

No cal que ens preocupem, de moment, pels conceptes d'herència i polimorfisme. En JavaScript, l'única herència és "formar part de", és a dir, el document forma part de la finestra, un formulari forma part del document i un camp de text forma part del formulari. Ens podem referir a qualsevol propietat d'un objecte utilitzant la sintaxi del punt, que ja hem vist:

```
finestra.document.formulari.camp.value
```

Pel que fa al polimorfisme, usar els mateixos noms per a propietats i mètodes similars en objectes diferents, només ens farà la programació més fàcil.

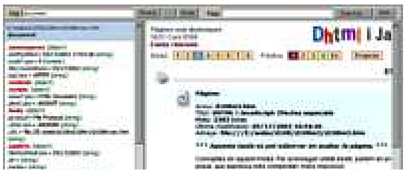
Window, document i tots els seus derivats no són els únics objectes que maneja JavaScript. N'hem vist d'altres, com ara Array, Math o String. De fet, tota l'estructura en basa en **object**, la "primitiva" que permet derivar els altres. Aleshores, quina és la diferència?

Podríem dir que hi ha dos grups o classes d'objectes: d'una banda, aquells que fan funcions internes del llenguatge, no tenen una representació a la pàgina i s'han de crear amb la declaració **new()**. D'altra banda, els objectes del DOM, que es corresponen amb elements de la pàgina i s'han de crear... amb HTML. També és cert que podem crear objectes DOM des de JavaScript, per exemple, ja hem utilitzat window.open(). En la majoria dels casos, però, ja són a la pàgina i JavaScript només els maneja.

Conceptes JavaScript en aquest capítol

La llista d'objectes, propietats i mètodes del DOM depèn del navegador. Tot i així, hi ha uns estàndards més o menys acceptats que funcionen en la majoria dels casos.

Aquest capítol és més teòric que no pas pràctic, de manera que hi ha molts conceptes. No hi són tots, i l'explicació de cadascun és molt breu. Presentar el DOM com cal requeriria tot un manual, així que, li farem una ullada i l'anirem coneixent millor la resta del curs.



En aquesta pràctica només hem d'observar les peces del DOM i, per això, utilitzarem l'inspector d'objectes, que ja hem vist en altres pràctiques.

En primer lloc, obrim una pàgina. Per defecte, al marc de l'inspector apareix l'objecte document. Els seus

objectes apareixen com a enllaç.

Si volem veure un altre objecte, l'escrivim al quadre Obj i fem clic en el botó Mostrar. També ho hem de fer si la pàgina del marc de la dreta canvia, per exemple, perquè hem fet clic en un enllaç.

Segons el navegador i la versió, ens podem trobar amb una llista curta o llarguíssima. A més, cada cop que surt una nova versió, s'afegeixen objectes, propietats i/o mètodes "segons la visió del fabricant", que poden no tenir equivalents als altres navegadors. La llista següent és, doncs, limitada i obsoleta, però només podia ser així.

1.- Window

Cada finestra del navegador, o cada "frame" en una estructura de marcs, és una instància de l'objecte window. Si, des de JavaScript, ens volem referir a la pròpia finestra o marc, podem utilitzar els noms **window** o **self**. Tot i que no és necessari, el codi queda més clar.

En canvi, si que és obligatori utilitzar un nom quan, des d'una finestra o marc, ens volem referir a una altra o als elements que conté. Si no ho féssim, s'entendria que volem actuar sobre la pròpia finestra o marc i, és clar, no funcionaria.

Ja hem vist algunes propietats, mètodes i objectes de window. Aquí en tenim un resum.

Objectes:

document: la pàgina continguda en la finestra o marc.
event: els events que es produeixen en l'àmbit de la finestra o marc.
frames: cadascun dels marcs, si n'hi ha.
history: llista d'adreces visitades i mètodes de desplaçament.
location: informació sobre l'adreça de la pàgina que mostra la finestra.
navigator: el navegador que maneja la finestra.
screen: propietats i mètodes de la pantalla.

Propietats:

closed: especifica que la finestra és tancada..
defaultStatus: missatge per defecte a la barra d'estat.
innerHeight, innerWidth: dimensions (vertical i horitzontal) de l'àrea de contingut.
length: número total de marcs (frames) que conté la finestra.
locationbar, menubar: barres d'adreces i menús.
name: el nom de la finestra, que ha de ser únic.
opener: si s'ha obert amb el mètode open, el nom de la finestra que ha obert l'actual.
outerHeight, outerWidth: dimensions, vertical i horitzontal, de la finestra.
parent: la finestra que conté a la finestra o marc actual.
scrollbars, statusbar: barres de desplaçament i d'estat.
status: missatge a la barra d'estat.
top: l'objecte window més alt en una estructura de marcs.

Mètodes:

alert: mostra un missatge d'alerta.
blur: treu el focus de la finestra o de l'objecte especificat.
captureEvents: prepara la finestra o el document per capturar un tipus d'event.
close: tanca la finestra especificada.
confirm: mostra un quadre de diàleg amb les opcions Acceptar i Cancel·lar.
find: cerca una cadena especificada en una finestra o marc.
focus: enfoca l'objecte especificat.
handleEvent: invoca la funció que controla un determinat event.
moveBy: mou la finestra una quantitat especificada en píxels.
moveTo: mou la finestra a unes coordenades de la pantalla (superior, esquerra).
open: obre una finestra nova.
print: imprimeix el contingut de la finestra o del marc.
prompt: mostra un quadre de diàleg amb un missatge i un camp d'entrada de dades.
releaseEvents: allibera els events capturats del tipus especificat.
resizeBy: redimensiona la finestra movent la cantonada inferior dreta una quantitat.
resizeTo: redimensiona la finestra a la mida especificada.
scroll: desplaça ta finestra a una coordenada especificada.
scrollBy: desplaça l'àrea de contingut una quantitat especificada.
scrollTo: desplaça el contingut d'una coordenada a la cantonada superior esquerra.
setHotKeys: activa o desactiva les tecles ràpides d'una finestra.
setInterval, setTimeout, clearInterval i clearTimeout: controlen els temporitzadors, i els

veurem detalladament en una altra pràctica.

2.- Document

L'objecte document és la pàgina, el seu contingut, les seves propietats i els mètodes per poder-hi enviar i controlar codi HTML. El navegador crea una instància de l'objecte document per cada etiqueta <body>. Cada finestra o marc té el seu objecte document.

Hem vist alguns dels seus objectes, propietats i mètodes. Ara en fem un resum.

Objectes:

all: tots els elements de l'objecte document. Només Internet Explorer.
anchors: col·lecció dels punts de fixació amb nom o identificador.
applets: els applets de Java del document.
body: el cos, entre les etiquetes <body> i </body>.
embeds: objectes "clavats" al document.
forms: Array que conté una entrada per a cada formulari de la pàgina.
frames: tots els objectes window presents al document.
images: un Array amb totes les imatges del document i les seves propietats.
layers: Array que conté una entrada per a cada capa de la pàgina. Només Netscape.
links: els enllaços de la pàgina.
plugins: un Array amb tots els plug-ins del document.
styleSheets: tots els fulls d'estils del document. També els enllaçats.

Propietats:

alinkColor: codi de color dels enllaços actius.
bgColor: codi de color del fons de la pàgina.
cookie: especifica una cookie, informació que es guarda a l'ordinador del client.
domain: nom del domini que ha servit la pàgina.
fgColor: codi de color del text.
fileSize: mida (pes) del document, en bytes. Només Internet Explorer.
formName: cada nom de formulari especificat al document.
height: altura del document, en píxels.
lastModified: cadena que especifica la data de l'última modificació de la pàgina.
linkColor: color dels enllaços.
title: títol de la pàgina.
URL: adreça completa del document.
vlinkColor: codi de color dels enllaços visitats.
width: amplada del document, en píxels.

Mètodes:

captureEvents: prepara la finestra o el document per capturar un tipus d'event.
close: tanca el document i l'obliga a mostrar-se.
getElementById: retorna l'objecte que té l'identificador especificat. Només versions 6
getElementsByName: retorna una col·lecció d'objectes amb el mateix nom. Versions 6
getSelection: retorna una cadena que conté el text actualment seleccionat a la pàgina.
handleEvent: invoca la funció que controla un determinat event.
open: obre un espai per recollir la sortida dels mètodes write o writeln.
releaseEvents: allibera els events capturats del tipus especificat.
write: escriu expressions HTML al document de la finestra especificada.
writeln: escriu al document i força un salt de línia en acabar.

Com a complement d'una llista que sembla que no s'hagi d'acabar mai, una funció que ensenya com podem modificar la propietat **document.bgColor** (atenció a la C majúscula) a partir d'uns botons que pot prémer l'usuari.

El formulari mostra molts més matisos (4096) que no pas els anomenats "colors segurs" (216), que utilitza el Dreamweaver. La funció resulta senzilla d'entendre i, en tot cas, ens pot resultar útil per provar noves combinacions.

Mireu el codi i proveu de contestar aquesta pregunta: com ho hem fet per aconseguir que els camps de text no siguin editables? (és a dir, que no s'hi pugui escriure). Una pista: l'operador especial **this**, situat en una etiqueta, especifica el propi objecte.

El DOM i el model d'events que manegen els diferents navegadors i versions són diferents. Això implica que els scripts han de tenir en compte aquestes diferències, si volem que funcionin en tots dos.

Els navegadors actuals permeten modificar gairebé tots els elements de la pàgina quan ja ha estat descarregada. Netscape 4.7, en canvi, és més restrictiu, com ho és també quan es tracta de fulls d'estils i, fins i tot, d'algunes etiquetes HTML.

No és estrany que Internet Explorer sigui el preferit a nivell mundial (75% d'usuaris, i creixent). Això, però, no vol dir que tothom l'utilitzi. El seu lligam amb Windows fa que els usuaris d'altres Sistemes hagin d'utilitzar Netscape o Mozilla, i no els podem pas oblidar. L'ideal és fer pàgines amb la màxima compatibilitat entre tots els exploradors.

En aquesta pràctica aprendrem a detectar el navegador i dirigir el flux de la pàgina en un o altre sentit , de manera que puguem superar, en la mesura que sigui possible, les greus incompatibilitats que els separen.

Conceptes JavaScript en aquest capítol

Objecte **navigator :** el navegador.
Propietat **appName :** mostra el nom del navegador.
Propietat **appVersion :** versió del navegador, en format text.
Propietat **platform :** la "plataforma" (sistema operatiu) del client.
Propietat **userAgent :** capçalera HTTP que informa sobre el navegador.
Mètode **javaEnabled() :** comprova si es poden executar applets de Java.
Objecte **document.all :** tots els elements de l'objecte document. Només Internet Explorer.
Objecte **document.layers :** Array que conté una entrada per a cada capa. Només Netscape.

1.- Navegador i versió

Hi ha molts navegadors i versions. Com que aquests programes són els responsables d'interpretar la pàgina i, entre ells, hi ha diferències notables, serà difícil que tot el que escrivim funcioni de la mateixa manera en totes les ocasions. De vegades haurem d'escriure porcions de codi o, fins i tot, pàgines senceres, específiques per a cada navegador.

Sortosament, JavaScript reconeix el navegador. Podem consultar les propietats de l'objecte **window.navigator** per saber amb quin estem tractant. Per veure-les, obrim qualsevol pàgina amb l'**inspector d'objectes**, escrivim "navigator" al quadre "Obj" i fem clic en el botó "Mostrar".

En aquest exemple només comprovarem dues propietats: el nom del navegador (**appName**) i la versió (**appVersion**). Aquesta última és un text i, en alguna ocasió, necessitarem discriminar a partir del número pelat. Cap problema, ho passem per la funció **parseInt()**.

El número de versió fa referència al nucli del navegador, de manera que si, per exemple, tenim Internet Explorer 6, la versió és 4, perquè les versions 4, 5 i 6 d'aquest navegador es basen en el mateix nucli, i aquest és el que interessa per saber com es comportaran les pàgines. Alguna vegada, però, necessitarem l'altre, perquè hi ha diferències importants.

```
<script language="javascript">
<!--
var navegador = window.navigator.appName;
var versio = window.navigator.appVersion;
document.writeln( 'Navegador: ' + navegador + '<br>Versió: ' + versió );
document.writeln( '<br>Versió del nucli: ' + parseInt( versió ) );
// -->
</script>
```

2.- Més informació

L'objecte navigator també proporciona altres informacions (té altres propietats) que, curiosament, depenen del navegador que utilitzem. Aquí sí, Netscape en té més, i algunes força interessants, com ara mimeTypes i plugins, dos Arrays que informen, respectivament, sobre els tipus d'arxius reconeguts pel sistema i els plug-ins que té instal·lats.

Pel que fa a les propietats "compatibles", que reconeixen tots dos navegadors, n'hi ha tres que en algun moment podem necessitar saber: plataforma, agent d'usuari i java activat.

Plataforma (**platform**) és el tipus de sistema operatiu que utilitza l'ordinador client. Els valors possibles són Min32, Win16, Mac68k, MacPPC i Unix. Naturalment, com que els sistemes i els navegadors evolucionen, aquest valors poden quedar obsolets en quatre dies.

I què vol dir "Agent d'usuari" (**userAgent**)? Quan el client envia una petició al servidor a través del protocol HTTP, s'acompanya d'algunes dades, és el que s'anomena "capçaleres HTTP". Aquesta, en concret, és la informació completa sobre el navegador que utilitza el client.

L'objecte *navigator* té un mètode que comprova si el navegador pot executar applets de java (**javaEnabled()**), molt útil abans de carregar-ne, doncs ho podem detectar i, en cas que no estigui activat, oferir una via alternativa.

En aquest exemple utilitzem una variable (javaOk) i l'operador condicional ternari (?) per tal de crear una frase a bocins: "Java " + (si Java activat, res, si no, "des") + "activat".

```
<script language="javascript">
<!--
var javaOk = navigator.javaEnabled();
document.writeln( 'Plataforma: ' + navigator.platform + '<br>' );
document.writeln( 'Agent d\'usuari: ' + navigator.userAgent + '<br>' );
document.writeln( 'Java ' + ( (javaOk) ? '': 'des' ) + 'activat' );
// -->
</script>
```

3.- Carregar arxius diferents

Quan no hi ha manera de fer compatibles els estils i/o les funcions entre tots dos navegadors, podem optar per mantenir arxius separats i utilitzar els uns o els altres. En aquest exemple s'enllaça un full d'estils i un script.

Si el navegador llegeix </script>, interpretarà que s'ha acabat javascript i ha tornat a HTML. Per tal d'evitar-ho utilitzem el truc, - barroer, però efectiu - d'escriure-ho a bocinets.

```
document.writeln( '</s" + "cript>');
```

Abans de decidir una solució com aquesta cal valorar, d'una banda, el fet de mantenir arxius separats i, de l'altra, el "pes" de carregar un únic arxiu amb tots els condicionals. En el primer cas, tenim l'inconvenient que, si fem modificacions, haurem de revisar dos arxius. En el segon ens estalviem aquesta feina, però la càrrega de la pàgina pot resultar més lenta.

```
<script language="javascript">
<!--
if ( navigator.appName.indexOf( "Microsoft" ) >= 0 ) {
  document.writeln( '<link rel="stylesheet" href="msie.css" type="text/css">' );
  document.write( '<script language="javascript" src="msie.js">' );
  document.writeln( '</s'+ 'cript>');
}
else {
  document.writeln( '<link rel="stylesheet" href="netscape.css" type="text/css">' );
  document.write( '<script language="javascript" src="netscape.js">' );
  document.writeln( '</s" + "cript>');
}
// -->
</script>
```

4.- Fes-ho, ... si ho pots fer

De vegades interessa saber si el navegador té una propietat específica o pot executar un mètode concret. Per exemple, quan tractem amb capes, Explorer i Netscape 4.7, a més de ser incompatibles, no les tracten de la mateixa manera en totes les versions.

Abans d'intentar usar aquestes propietats o mètodes cal, doncs, comprovar què es pot fer. Si coneixem totes les peculiaritats de les diverses versions dels navegadors podrem usar la tècnica de l'exemple anterior. Gairebé sempre, però, resulta més pràctic, curt i segur comprovar directament la disponibilitat d'allò que volem utilitzar.

Internet Explorer i els altres navegadors actuals tenen **document.all**, un Array amb tots els components de la pàgina. Si un element no pertany a cap col·lecció (p.ex. no és cap imatge, formulari, marc, ...) però li hem posat un identificador, el tindrem a l'abast a document.all[*identificador*]..

Netscape 4.7, en canvi, té la col·lecció **document.layers**. Es tracta d'un Array amb totes les capes definides. Els seus elements no són accessibles directament (són objectes), però sí que ho són les propietats, p.ex. document.layers[*identificador*].bgColor.

> La versió 2 del DOM, implementada a Netscape 6, a Mozilla i, qui sap, potser un futur estàndard, aporta el mètode **document.getElementById**, que permet referir-se als objectes pel seu identificador. En aquest exemple, però, no l'utilitzarem.

Aquí volem utilitzar aquesta diferència per dirigir el flux de la pàgina, segons el navegador. Si té document.all, és l'Explorer, i si té document.layers, és Netscape 4.0 o superior. Si ho hem de comprovar diverses vegades és millor que ho guardem en una variable de tipus lògic:

```
var ns = document.layers;
if ( ns )      ... instruccions Netscape 4.7 ...
else           ... instruccions iExplorer ...
```

L'exemple crea "rollovers" de text, enllaços que, quan passa el ratolí, canvien el seu color de fons. Farem els vincles sobre contenidors (span), amb identificador únic i posició relativa.

Assignarem els events onMouseOver i onMouseOut a la funció "pinta", de manera que, quan passi el ratolí, s'activi un color de fons i, quan surti, es desactivi. A més, utilitzarem colors diferents i funcionarà amb Netscape 4.7, tot un luxe i una alternativa al **a:hover** de css.

La funció rebrà dos paràmetres: l'identificador i el color. Si s'utilitza amb Netscape 4.7, primer comprovarà que el color no estigui buit i, si és així, li assignarà el valor **null**. A continuació modificarà el color de fons del contenidor, **document.layers[*idSpan*].bgColor**. Si utilitzem Éxplorer, la sintaxi és diferent: **document.all[*idSpan*].style.background**.

Les propietats **all** i **layers** només són un petit exemple del que es pot fer. Si observem qualsevol pàgina amb l'inspector d'objectes, amb tots dos navegadors, veurem que tenen propietats ben diferents. Així podem saber què funcionarà i, alhora, com els podem distingir.

A la pràctica següent veurem més detalls sobre identificadors, posicions... .Tornen els estils!

```
<script language="javascript">
<!--
var ns = document.layers;
function pinta( idSpan, clr ) {
  if (ns){
    if (clr=='') clr = null;
    document.layers[idSpan].bgColor = clr;
  }
  else document.all[idSpan].style.background = clr;
}
// -->
</script>
</head>
```

```
<body><h3>
<a href="pagina1.htm"
onMouseOver="pinta( 'lnk1', '#CCEEBB' )" onMouseOut="pinta( 'lnk1', '' )">
<span id='lnk1' style="position:relative">Enllaç 1</span></a>
<a href="pagina2.htm"
onMouseOver="pinta( 'lnk2', '#FFDDBB' )" onMouseOut="pinta( 'lnk2', '' )">
<span id='lnk2' style="position:relative">Enllaç 2</span></a>
<a href="pagina3.htm"
onMouseOver="pinta( 'lnk3', '#CCDDFF' )" onMouseOut="pinta( 'lnk3', '' )">
<span id='lnk3' style="position:relative">Enllaç 3</span></a></h3>
...
```

En aquesta pràctica veurem una introducció al concepte d'estils.
En el mòdul 1, pràctica 3 ja s'ha fet una introducció als estils CSS, en aquell cas, des de l'aplicació amb el Dreamweaver.

Amb el que s'explica a continuació es pretén donar una visió més
global del concepte d'estils i les seves formes d'aplicació, tant en temps de disseny com en temps de visualització.

Abans de començar hem de tenir molt clar que les directives d'HTML són objectes que actuen de contenidors d'informació. Aquests "contenidors" poden contenir, a la vegada, d'altres directives (contenidors) i informació.

S'estableix d'aquesta forma una jerarquia d'objectes i per tant una herència (les directives més enniuades hereten propietats de les directives que les contenen). Aquesta última afirmació té, però, excepcions, poques, en algunes versions del navegador Netscape que són una de les principals fonts d'incompatibilitat que cal detectar.

Per defecte, les directives apliquen al seu contingut els aspectes propis de la directiva que ha definit el dissenyador del programa navegador (Netscape i Microsoft). Així, per exemple, la directiva ... és un objecte contenidor de dades que hereta les propietats d'altres directives (objectes) que la contenen i a més afegeix la propietat de negreta.

Fixeu-vos que estem parlant de propietats que s'apliquen al contingut de les directives. Fins l'aparició dels Fulls d'estil en cascada (CSS) no teníem l'opció de modificar aquestes propietats que aplica cada contenidor, tret de les que podíem manipular mitjançant els paràmetres d'algunes directives.

Una bona manera d'aprendre el funcionament dels estils i veure totes les seves possibilitats, és mitjançant el Dreamweaver. Podem aplicar estils CSS utilitzant la seva interfície i posteriorment visualitzar el codi que ha generat. També disposem d'una ajuda en línia que ens mostra totes les propietats disponibles i les seves compatibilitats entre versions dels navegadors a "Ayuda/Referencia" o prement "Majúscules + F1".

En aquesta pràctica es pretén veure com podem modificar les propietats de les directives (objectes) HTML, en temps de disseny i en temps de visualització mitjançant els estils CSS i JavaScript.

També es comentarà una llista resumida dels principals estils utilitzables per modificar propietats de text, colors i fons. Altres tipus de propietats CSS es deixen per més endavant.

Elements CSS en aquest capítol.

Paràmetre **Style** : Paràmetre per modificar propietats CSS de les directives HTML.
Propietat **background-image** : Defineix la imatge de fons.
Valor CSS **url(url_imatge)** : assigna una url a una propietat.
Directives **<DIV>** i **<SPAN>** : Directives HTML.
Clases **a :hover,a :active,a :link,a:visited** : classes per modificar el comportament dels links.
Directiva **<LINK ...>** : Carrega un fitxer .css amb definicions d'estils.

1.- Aplicació dels estils.

Hi han diverses maneres d'aplicar estils a les diferents directives d'HTML. La més elemental consisteix a afegir un nou paràmetre de nom "Style" a la directiva en què desitgem modificar una propietat. Qualsevol directiva d'HTML contenidora d'informació i/o altres directives admet aquest nou paràmetre. (Tingueu en compte que gairebé totes les directives tenen propietats CSS modificables).

La forma d'aplicar valors a les propietats CSS és fa amb el parell: **propietat : valor** . Noteu com s'utilitzen els : per a fer l'assignació. Múltiples parells "propietat : valor" es poden encadenar mitjançant el punt i coma ; . Un exemple d'ús podria ser:

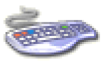
```
<directiva style="propietat:valor; propietat:valor; .....">........
```

Si aquesta directiva contingués d'altres directives, aquestes últimes també heretarien els valors de les propietats indicades, tret d'algunes excepcions que s'han de comprovar amb els diferents navegadors.

Directives <DIV> i <SPAN>
Aquestes dues directives HTML són les contenidores per excel·lència. S'utilitzen per definir àrees i blocs d'informació, sobre les quals s'hi poden afegir estils CSS i referenciar-les des de JavaScript. Elles, per si soles, no afegeixen cap propietat nova al seu contingut. Però <div> inclou un salt de línia, abans i després i <span> no. Habitualment <div> es fa anar per blocs de dades més grans i <span> per blocs de dades més petits i enniuats dins de blocs <div>.

Les utilitzarem en aquesta pràctica per aplicar les diferents propietats dels estils CSS. Més endavant, també les utilitzarem per definir capes.

El següent exemple ens mostra com modifiquem algunes propietats de directives HTML que fan que el seu comportament sigui totalment diferent del que estem acostumats a veure.



```
<body>
  <b style="Font-weight:normal; Font-style:italic">
    Hauria de ser negreta, en canvi no ho és
  </b><br>

  <font color="red"><p style="color:blue">
    Hauria d'esser vermell, però no ho és.
  </p></font><br>

  <p align="left" style="text-align:right">
    Això no hauria d'estar a l'esquerra?
  </p>
</body>
```

2.- Algunes propietats modificables amb CSS.

En aquest apartat es farà un resum de les principals propietats CSS classificades dins el seu àmbit d'actuació. No hi són totes, però, si que hi són les més habituals. Algunes no són compatibles amb tots els navegadors i versions. Recordeu que podeu trobar una guia més extensa i referències sobre compatibilitat a l'ajuda del Dreamweaver (Ayuda/Referencia).

Unitats de mesura.
Les unitats de mesura es precedeixen opcionalment amb el signe + o - i finalitzen amb el símbol de la unitat. Aquestes unitats de longitud poden ser:

em	Un em és igual a la mida de la font actual.
ex	Un ex és igual a la mitjana de l'alçada de la font actual. Habitualment, l'alçada de la lletra x.
px	píxels
in	polsades
cm	centímetres
mm	mil·límetres
pt	punts (1 pt = 1/72 polsada)
pc	pica (1 pc = 12 pt)
%	percentatge corresponent a la mida de la font actual.

Colors:
Per als colors podem utilitzar les paraules bàsiques disponibles: blue, red, cyan, green,

yellow, etc. o bé els colors RGB:

#rrggbb	Exemple: #335a4c
#rgb	Exemple: #03a
rgb(n,n,n)	on n és un núm. entre 0 i 255. Exemple: rgb(80, 245, 135)
rgb(%, %, %)	on % és el percentatge de color. Exemple: rgb(34%, 80%, 100%)

Propietats de tipus de lletra:

Font-size:	Mida de la lletra. *small*, *medium*, *large*, o valor enter o percentatge (%).
Font-weight:	Gruix de la lletra. *Normal*, *bold*, *bolder*, *lighter* o un valor enter.
Font-style:	Estil del tipus de lletra. *Normal*, *itàlic*, *oblique*.
Font-family:	Família genèrica de tipus de lletra. *Serif*, *sans-serif*, *cursive*, *fantasy*, *monoespace*,
Font:	Propietat genèrica que defineix totes les anteriors d'un sol cop. L'ordre a indicar les diferents propietats és el següent: Font-family, font-weight, font-style, font-size.

Propietats de color i fons:

Color:	Color de l'element Html.
Background-color:	Color de fons d'un element Html
Background-image:	Imatge de fons d'un element Html.

Propietats de text:

Word-spacing:	Indica la longitud de l'espai entre paraules
Letter-spacing:	Indica la longitud de l'espai entre lletres.
Text-decoration:	Mostra efectes de text. Pot valer: *none*, *underline*, *overline*, *line-through* i *blink*.
Vertical-align:	Indica l'alineació vertical, pot ser un percentatge o les paraules: *baseline*, *middle*, *sub*, *super*, *text-top*, *text-bottom*, *top* i *bottom*.
Text-transform:	Indica efectes de text. Pot valer: *capitalize*, *uppercase*, *lowercase*, *none*.
Text-align:	Indica l'alineació del text. Pot valer: *left*, *right*, *center* i *justify*.
Text-indent:	Indica la indentació del text. Pot valer una longitud o un percentatge.

Propietats de quadre:

Margin-left:	Especifiquen els marges esquerra, dret, superior i inferior del document respectivament, la propietat margin els engloba a tots en conjunt. Exemple: Margin-left: 20px; Margin-right: 3.5em; Margin-top:10%; Margin-bottom: 15px . Tots ells es podrien haver englobat amb Margin: 20px 3.5em 10% 15px.
Margin-right:	
Margin-top:	
Margin-bottom:	
Margin:	
Padding-left:	Especifiquen l'espai esquerra, dret, superior i inferior entre el contingut d'un element i el marge, respectivament. La propietat Padding: els engloba a tots en conjunt. Exemple: Padding-left: 20px; Padding-right: 3.5 em; Padding-top: 10%; Padding-bottom: 15px. Tots ells es poden englobar amb Padding.
Padding-right:	
Padding-top:	
Padding-bottom:	
Padding:	
Border-style:	Indica l'estil d'un marge. Pot valer: *none*, *dotted*, *dashed*, *solid*, *double*, *groove*, *ridge*, *inset* i *outset*.
Border-width:	Indica l'ample del marge d'un element. Pot valer: *thin*, *medium*, *thick* o una longitud.
Border-color:	Especifica el color d'un marge. Es pot indicar entre 1 i 4 valors.
Border	Determina l'ample, l'estil i el color del marge d'un element en l'ordre indicat.

3.- Altres formes d'incloure estils a la web.

A l'apartat 1 hem vist com manipulem els estils dels contenidors HTML mitjançant el paràmetre style. Aquest és un mètode prou vàlid, però, no és gaire eficaç, sobretot si desitgem modificar els estils de múltiples directives. En aquest apartat veurem les formes més habituals de definir els estils, aquestes formes són les recomanades pel W3C.

Existeixen tres mètodes de definir estils globalment: les **classes**, la **redefinició de directives** i els **identificadors**. També comentarem com podem introduir tots els estils dins un fitxer .css i carregar-lo en una o vàries pàgines a la vegada. Totes aquestes opcions s'introdueixen dins el bloc <head> entre les directives <style> i </style>.

Mitjançant /* …. */ podem introduir comentaris dins les definicions de les propietats CSS.

Redefinició de directives.
Aquesta opció permet redefinir el comportament de les directives de forma global dins tot el document. S'ha d'indicar el nom de la directiva seguida de **{ }**. Dins les claus s'indiquen els parells **propietat : valor** separats amb un ; . Els valors dels paràmetres indicats afectaran totes les directives del tipus indicat dins del document HTML. Per exemple:

```
<style type="text/css">
  P {font-size: large; font-weight:bold}
  H1 U {color:#ff0000} /*introdueix el color vermell*/
  H2, A {font-style: italic; background-color : #00ff00}
</style>
```

La línia començada per P, defineix per al contingut de totes les directives <p> del document una mida de font igual a *large* i un gruix de font igual a *bold*.

La línia començada per H1 defineix per al contingut de totes les directives <u> incloses dins de directives <h1> el color de text vermell (fixem-nos que en aquest cas no hi ha una coma per separar a H1 i U i això implica un comportament molt diferent al que s'explica a continuació).

La línia començada per H2 defineix per al contingut de totes les directives <h2> i <a> un tipus de font itàlica i un color de fons verd (aquí les dues directives estan separades per coma, el comportament és diferent en aquest cas).

Les classes.
Podem indicar un conjunt de propietats i assignar-los un nom de classe. El nom de classe es situarà a continuació del nom de la directiva unida per un punt: **directiva.nom_classe { ….. }**. Això ens permet definir propietats diferents per a una mateixa directiva. Dins la directiva corresponent en el bloc <body>, s'haurà d'indicar la classe que desitgem utilitzar mitjançant el paràmetre "class=….".

També podem definir **classes genèriques**, és a dir, que no estan associades inicialment a cap directiva. Aquestes s'indiquen solament amb el nom de la classe amb un punt al davant. **.nom_classe_genèrica { …. }**. La classe genèrica la podem associar a qualsevol directiva mitjançant el paràmetre "class= … ". En el següent exemple:

```
<style type="text/css">
  P.groc {color:#ffff00}
  P.blaufluix {color:#00ffff}
  .lila {color : #A801FF}
</style>
```

Les dues línies començades per P defineixen dues classes aplicables a directives <p> que introdueixen els colors groc i blau fluix al seu contingut. A diferència de la redefinició d'etiquetes, això ens permetrà elegir el color a aplicar per qualsevol directiva tipus P.

La línia començada per .lila defineix una classe genèrica aplicable a qualsevol directiva que introduirà el color lila al seu contingut.

Un cop definides les classes dins el bloc <head>, les podem aplicar a les directives corresponents dins el bloc <body> de la següent forma:

```
<body>
…
<p class="blaufluix"> Això es veurà de color blau fluix </p>
…
<h2 class="lila"> Això es veurà de color lila </h2>
…
<p class="groc"> Això es veurà de color groc </p>
…
```

```
</body>
```

Existeixen també en l'estàndard CSS les anomenades **classes falses**. Aquestes detecten l'estat d'una directiva <a ...> i en funció de l'estat apliquen uns estils o no. No totes són reconegudes per Netscape. Aquestes són :

- **:link** - Representa qualsevol element que es comporti com un enllaç
- **:visited** - Representa un enllaç visitat.
- **:hover** - Representa un enllaç assenyalat.
- **:active** - Representa un enllaç amb el focus activat.

La forma habitual d'utilitzar aquestes classes és per modificar l'aspecte dels enllaços:

```
<style type="text/css">
  a:hover { color:#ffff00}
  a:visited {color:#ff0000}
  a:link {color:#00ff00}
  a:active {color:#0000ff}
</style>
```

Els identificadors.
Els identificadors serveixen per identificar una directiva mitjançant un nom específic i únic. Per donar-li un nom a una determinada directiva s'utilitza el paràmetre **id="nom_identificador**". No pot haver més d'una directiva dins un document HTML amb el mateix identificador. Això permet referenciar la directiva des d'un altre lloc del document.

La utilitat de l'identificador també s'estén a altres opcions de DHTML que es veuran a l'apartat següent.

Aquesta opció ens permet indicar propietats CSS d'una determinada directiva fora del lloc que aquesta ocupa . Podríem dir que equival al que s'ha explicat en l'apartat 1, però, si el nombre de propietats CSS és elevat quedarà més bé si aquestes estan situades en el bloc <head>.

La seva utilització es veu en el següent exemple:

```
<head>
…
<style type="text/css">
#ident1 {margin-left : 150px; margin-right : 100 px; color : #0000ff; font-weight : bold}
#ident2 {color:yellow; font-size:12 pt; text-align:justify; background-image:url(/img
/fons.gif)}
</style>
…
</head>

<body>
…
<p id=ident1> Text entre marges de color blau i negreta </p>
<div id=ident2> Àrea amb una imatge de fons i text justificat de color groc </div>
…
</body>
```

Les diferents formes de definir estils que s'han comentat en aquest apartat es poden incloure dins un fitxer de text amb extensió .css. Posteriorment, es pot carregar el contingut d'aquest fitxer a la pàgina desitjada mitjançant la següent directiva HTML situada al bloc <head> :

```
<link rel="stylesheet" href="fitxer.css" type="text/css">
```

D'aquesta forma, s'aconsegueix tenir una única definició d'estils que es pot aplicar a múltiples documents HTML tot donant un aspecte uniforme per a totes les pàgines i, a més, fer que siguin fàcilment modificables, ja que la definició es troba en un únic fitxer.

Quan existeixen múltiples assignacions sobre una mateixa propietat CSS, l'única que prevaldrà serà l'última assignada.

En el següent exemple es pot veure com es defineixen diversos estils en el bloc <head> i

s'apliquen a diferents directives dins el bloc <body>. De les definicions efectuades dins la directiva <style> corresponen les tres primeres a redefinició de directives, les cinc següents a classes i l'última a definició mitjançant identificador.

En el bloc <head>:

```
...
<style type="text/css">
    body {Background-color: #cad5fa}
    td b {color:rgb(100%,0%,0%)} /*color vermell*/
    i,u {letter-spacing:4px}

    h3.tcap {text-transform:capitalize}
    h3.tindent {text-indent:5em}
    .blau {color: #0000ff; border:medium groove #00ffff}
    .marro {color: #6A3A3A; background-color:#3A6A60}
    .caixa {background-image:url(fons.gif);color:blue}

    #ident1 {color:#ff0000}
</style>
...
```

En el bloc <body>:

```
...
Text sense cap estil aplicat.
<table border=1>
    <tr><td><b>Text afectat per la redefinició "td b".</b></td>
    <td>Text sense cap estil aplicat.</td></tr>
</table>
<form>
    <input type="text" size=30 value='Afectat per la clase "caixa" ' class="caixa">
    <br><br>
    <textarea rows=4 cols=25 class="caixa">
        Contingut afectat per la clase "caixa"
    </textarea>
</form>
<b>No afectat per la redefinició "td b"</b><br>
<i>Afectat per l'estil redefinit per "i, u"</i><br>
<u>Afectat per l'estil redefinit per "i, u"</u>
<h3>No afectat per la redefinició de h3</h3>
<h3 class=tcap>afectat per la classe "h3.tcap"</h3>
<h3 class=tindent>Afectat per la classe "h3.tindent"</h3>
<div class=blau>Afectat per la classe genèrica ".blau"</div>
<span class=marro>Afectat per la classe genèrica ".marro"</span>
<div id=ident1>Afectat per l'identificador d'estil "#ident1"</div>
...
```

4.- Contenidors i identificadors. Manipulació amb JavaScript.

Els estils CSS són un gran avenç a l'hora de definir les propietats dels documents HTML de forma fàcil, però, en tot el que hem vist fins ara, aquestes propietats s'han aplicat en temps de disseny. És a dir, hem modificat propietats CSS mentre dissenyem la pàgina, però un cop s'està visualitzant el resultat ja no és possible tornar-lo a canviar.

JavaScript inclou la possibilitat de modificar aquestes propietats en temps de visualització, és a dir, mentre s'està visualitzant la pàgina. Tal com ja s'ha dit des del principi, les directives HTML són objectes que tenen propietats. Per tant, podem fer referència a qualsevol d'aquests objectes (directives) i modificar les seves propietats.

Ara bé, en aquest cas ens trobarem amb greus dificultats de compatibilitat entre els navegadors de Microsoft i Netscape. Aquest últim no permet accedir a determinades propietats en temps d'execució, per tant, els exemples que vénen a continuació no funcionen totalment per Netscape.

Per poder-ho fer, és imprescindible que cada directiva a modificar tingui un identificador, el mateix que ja s'ha indicat en l'apartat 3 referent a l'aplicació d'estils mitjançant identificador. Un cop tenim aquest identificador ja el podem utilitzar com un objecte dins el programa JavaScript.

Si dins el codi HTML tenim una directiva d'aquests tipus:

```
<div id=ident1 style="color:blue"> text en color blau, i minúscules </div>
```

Dins un programa JavaScript podríem fer referència a aquesta directiva i canviar-li alguna propietat. En l' iExplorer existeix l'objecte **document.all** que conté a totes les referències que existeixen dins el bloc *<body>*. Per exemple, les següents línies de codi:

```
document.all.ident1.style.color="red";
document.all.ident1.style.textTransform="uppercase";
```

Canviarien el color de la frase a vermell i totes les lletres passarien a majúscules. IE també permet referir-nos als identificadors sense utilitzar el text *document.all*, tal com veurem en l'exemple que hi ha en aquest apartat.

Noteu que en aquest cas, ja no s'utilitza els : per assignar un nou valor a la propietat CSS i a més, les propietats que inclouen un guionet en el seu nom, aquest desapareix i s'utilitza el nom sense guionet i la primera lletra de la segona paraula en majúscula, tal com es pot veure a la propietat text-transform.

En el següent exemple podem veure com canviem aquestes propietats amb JavaScript. Aquest exemple utilitza dues directives <div> per ubicar dins la pàgina dos textos iguals. Algunes lletres dels textos també estan incloses dins de directives <span> per poder-les referenciar de forma particular. Totes aquestes directives estan referenciades per un identificador.

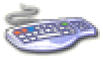
L'efecte produït consisteix en la frase "DHTML i Javascript", el primer <div> correspon al text superior i el segon <div> correspon al text inferior que representa l'ombra. Noteu com, dins la definició d'estils situada dins de *<head>*, les propietats **top** i **left** de les etiquetes **sup** i **inf** estan desfasades 5 punts per efectuar l'efecte d'ombra.

L'identificador de les directives <div> i <span> servirà per poder-les referenciar des de la definició d'estils i des de la funció JavaScript **cambia()**, manipulant d'aquesta forma les seves propietats. Aquesta funció s'inicia per primer cop en carregar-se la pàgina mitjançant el controlador d'event **onLoad** situat dins la directiva *<body>*. Posteriorment, la funció es repeteix periòdicament per l'acció del temporitzador activat amb el **setTimeout()**.

Al bloc <head>, dins la directiva <style> s'indiquen els estils inicials a aplicar a cadascun dels contenidors <div> i <span>. L'efecte que es visualitza és el d'un text amb lletres de colors i una ombra de fons.

La funció **cambia()** alterna els colors del text "Dhtml" de forma rotativa llegint i assignant els valors de les propietats **color** i modifica les posicions de les lletres "h" i "m" manipulant la seva propietat **vertical-align**.

La funció *cambia()* solament s'executa per al navegador IE (detecta l'objecte *document.all*), en el cas de tractar-se d'un navegador NN4 es mostra un text alternatiu i si és un navegador NN6 es mostra la frase sense animació. Si voleu aprofundir una mica més pel que fa a la compatibilització entre IE i NN6 podeu analitzar el codi per intercanviar els colors de les lletres CSS situades al principi d'aquesta pàgina, veureu també, com s'introdueix el text alternatiu per a NN4.



```
<html>
<head>
<title>Exemple</title>

<style type="text/css">
  .comu {position:absolute; font-size:50px; font-family:impact}
  #sup {top:20pt; left:20pt; z-index:1; color:#405048}
  #inf {top:25pt; left:25pt; z-index:0; color:#B8B8B8}
</style>

<script language="JavaScript">
function cambia()
{
  if (document.all) { //detectem que correspongui a un navegador IE.

    //efectuem la alternancia de colors en les lletres DHTML
    var colorID = lD.style.color;
    lD.style.color = lh.style.color;
```

```
    lh.style.color = lt.style.color;
    lt.style.color = lm.style.color;
    lm.style.color = ll.style.color;
    ll.style.color = colorlD;

    //efectuem la alternancia de posició vertical de les lletres h i m i les seves ombres.
    if(lh.style.verticalAlign=="sub") {
        lh.style.verticalAlign="baseline";
        lh2.style.verticalAlign="baseline";
        lm.style.verticalAlign="sub";
        lm2.style.verticalAlign="sub";
    }
    else {
        lh.style.verticalAlign="sub";
        lh2.style.verticalAlign="sub";
        lm.style.verticalAlign="baseline";
        lm2.style.verticalAlign="baseline";
    }
    setTimeout("cambia()", 1000); //repetim la funció al cap d'1 s.
  }
}
</script>

</head>

<body onLoad="cambia();">

<div class="comu" id="sup"> <!-- Text en color, en primer pla -->
    <span id="lD" style="color:#0070E0">D</span>
                                          <span id="lh" style="color:#FF0000">h</span>
                                          <span id="lt" style="color:#009000">t</span>
                                          <span id="lm" style="color:#9000D8">m</span>
                                          <span id="ll" style="color:#FF8000">l</span> i Javascript
</div>

<div class="comu" id="inf"> <!-- Text d'ombra en segon pla -->
    D<span id="lh2">h</span>t<span id="lm2">m</span>l i Javascript
</div>

</body>
</html>
```

Atenció a la diversitat > Control de l'entorn

Quan dissenyem una pàgina, l'ideal seria que ocupés tota l'àrea disponible i mostrés els colors correctament. Com que, d'entrada, no sabem quina mida tindrà la pantalla, i a més, esperem tenir molts visitants amb monitors de tot tipus, procurem fer un disseny que s'adapti a la majoria de configuracions.

Amb JavaScript podem saber les mesures de la pantalla i la profunditat del color, és a dir, quants bits utilitza la targeta per representar cada píxel. A partir d'aquesta informació podem fer que la pàgina prengui decisions.

L'objecte window guarda aquestes informacions, i d'altres, a l'objecte **screen**. En aquest cas, però, no utilitzarem mai la sintaxi window.screen perquè, per a qualsevol finestra, la pantalla és sempre la mateixa. La sintaxi serà, doncs, només **screen.*propietat***.

En aquesta pràctica aprendrem a detectar la configuració i a usar les dades de l'objecte screen. També aprendrem a treure la pàgina dels marcs, i estudiarem com fer pàgines que no deixin rastre i com camuflar l'adreça de correu.

Conceptes JavaScript en aquest capítol

Objecte **screen :** Propietats de la pantalla (en realitat, de la targeta gràfica). No té mètodes.
Propietats **width** i **height :** amplada i altura total de la pantalla.
Propietats **availWidth** i **availHeight :** amplada i altura que pot ocupar una finestra.
Propietats **colorDepth** i **pixelDepth :** profunditat de la paleta i dels píxels, en bits.
Mètode **Math.pow() :** fa una potència (base elevada a un exponent).
Mètodes de window, **moveTo()** i **sizeTo() :** mouen i dimensionen la finestra.
Instàncies de window, **parent** i **top :** la finestra anterior i la més alta en la jerarquia.
Mètode **location.replace() :** reemplaça l'adreça actual per la que rep com a paràmetre.

1.- Resolució i colors

L'objecte **screen** no té instàncies ni mètodes. No podem fer res amb la pantalla des d'una pàgina web. És més, seria perillós que es pogués fer perquè, aleshores, qualsevol pàgina d'Internet podria desconfigurar-nos l'ordinador i posar-nos en un bon embolic. Què passaria si, per exemple, es forcés una configuració que la màquina no pot admetre?

Les propietats de l'objecte screen només informen. Això sí, podem utilitzar les dades per crear finestres o modificar-ne la mida o posició. I, si hem d'usar condicionals, també ens podem plantejar pàgines senceres que s'adaptin als diferents entorns.

En aquest exemple mostrarem la resolució i els colors disponibles. Utilitzarem les propietats **screen.width** i **screen.height** que ens donen, respectivament, l'amplada i l'altura totals de la pantalla. Per als colors, utilitzarem **screen.colorDepth** que ens diu quants bits utilitza la paleta per definir cada color. Si, com és normal avui en dia, la targeta gràfica no es basa en una gamma de colors, es mostra la "profunditat" del píxel (pixelDepth) també en bits.

Aquestes són propietats que només podem llegir (read only). No es poden modificar.

Per saber quants colors mostra la tarja gràfica, fem una potència amb el número 2 com a base i **colorDepth** com a exponent. Per això, utilitzem la funció **pow** de l'objecte **Math**. Ho escriurem tot en una finestra nova que s'obrirà quan fem clic en un vincle.

```
<script language="javascript">
<!--
function pantalla() {
  window.focus();
  var fn = window.open( '', 'win1', 'width=360,height=100,top=260,left=200' );
```

```
  with( fn.document ) {
    writeln( 'Resolució de pantalla: <b>' + screen.width );
    writeln( 'x ' + screen.height + '</b><br>' );
    writeln( 'Profunditat de color: <b>' + screen.colorDepth + ' bits,' );
    writeln( Math.pow( 2, screen.colorDepth ) + ' colors</b><br>' );
    close();
  }
  fn.focus();
  return false;
}
// -->
</script>
</head>

<body ...>
<a href="" onClick="return( pantalla() )">Propietats de la pantalla</a><br>
...
```

2.- Ocupar tot l'espai disponible

Des d'una pàgina no podem crear finestres maximitzades ni maximitzar la finestra, això ho ha de fer el sistema operatiu. Si utilitzem Internet Explorer podem fer que la pàgina ocupi tota la pantalla quan creem una finestra, escrivint **fullscreen=1** a les seves propietats:

```
var fnestra = window.open( 'arxiu.htm', 'nom', 'fullscreen=1' )
```

De vegades ens pot interessar aquest efecte tot i que, quan el trobem en una altra web, no ens fa massa gràcia. Maximitzar és una altra cosa: és tenir la finestra, amb tots els seus elements (menús, botons,...) ocupant tot l'espai disponible i sense "bordes".

No ho podem fer ... del tot. Es pot ocupar l'espai màxim, però no ens lliurarem dels "bordes". A més, haurem de seguir estratègies diferents per als dos navegadors.

Primer, movem la finestra a l'angle superior esquerre: **window.moveTo(0,0)**. A continuació, declarem dues variables (x,y) que contindran l'amplada i alçada de l'àrea disponible per a finestres. En aquesta mesura ja s'ha descomptat l'espai ocupat pels elements fixos del sistema operatiu, p.ex. la barra de feines, del Windows.

var x = **screen.availWidth**
var y = **screen.availHeight**

A partir d'aquí, hem de detectar el navegador. Si és un navegador actual, només cal utilitzar el mètode **resizeTo(x,y)** de l'objecte window. Si provem el mateix amb Netscape 4.7, la finestra es farà més gran que la pantalla. Hem d'utilitzar un altre mètode.

Netscape 4.7 reconeix les propietats **outerWidth** i **outerHeight** de la finestra, que indiquen l'amplada i l'alçada (iExplorer no les accepta). Comparem si alguna de les dues és menor del que poden ser. Utilitzem l'operador "o lògic" (**||**):

Si l'amplada de la finestra és menor que "x" ***o*** *l'altura és menor que* "y" ...

Aleshores assignem "x" a l'amplada i "y" a l'altura. El navegador redimensionarà la finestra i tornarà a "pintar" la pàgina. Probablement ens quedarà més clar si llegim el codi del script que no pas amb l'explicació. És molt senzill.

```
<script language="javascript">
window.moveTo(0,0)
var x = screen.availWidth
var y = screen.availHeight
if ( document.all ) {
  window.resizeTo( x,y )
}
else {
  if ( window.outerWidth<x || window.outerHeight<y ){
    window.outerWidth = x
    window.outerHeight = y
  }
}
document.writeln( '<h1>Tot l\'espai és meu!</h1>' )
</script>
```

3.- Sortir de la gàbia

Hem dissenyat una pàgina amb cura, ha quedat bé i algú l'enllaça des de la seva web. Com queden les mides i els elements? Si la nostra pàgina s'obre a la mateixa finestra, perfecte, tot i que no és habitual que algú convidi els visitants a abandonar el seu lloc per anar al nostre. El més probable és que ens l'obrin en una finestra o en un marc.

Si la pàgina s'obre en una nova finestra, en podem controlar, relativament, la mida, tal com hem vist a l'exemple anterior. Si s'obre en un marc ... però, com sabem que som en un marc?

Un marc (frame) és una finestra en una finestra. El marc, un objecte, és fill d'un altre objecte, i sap com es diu el seu pare (ja passa, això). El pare (**parent**) té una adreça i el marc-fill en té una altra. El pare pot ser, alhora, fill. De tota la saga, el primer es **top**. En aquest cas, i només en aquest, l'adreça és la mateixa que la de son pare, perquè no en té.

Si l'adreça del pare no és la meva, aleshores, vull la del pare !!!

Això és exactament el que farem, comprovar que l'adreça de la finestra de la nostra pàgina és la mateixa que la del contenidor. Si no és així, segur que hem caigut en un frame. Aleshores assignem la nostra adreça a la del pare. Com que la pàgina es tornarà a carregar, es torna a fer la comprovació i, si encara no hem arribat al "top", repetim.

És clar que també podem anar directament a dalt: posem la nostra adreça a **top.location.href** i ja hi som. Aquí ho fem de les dues maneres. La segona compara les finestres, no pas les adreces. En tot cas, aquestes instruccions han d'anar al principi de la pàgina.

Mètode "pas a pas"

```
<script language="javascript">
<!--
if ( parent.location.href != window.location.href ) {
  parent.location.href = window.location.href;
}
// -->
</script>
```

Mètode directe

```
<script language="javascript">
<!--
if (window != window.top) {
  top.location.href = location.href;
}
// -->
</script>
```

4.- La web invisible (pastilles anti-paranoia)

L'exemple anterior ens hauria de demostrar que, amb JavaScript i, en general, quan es programa, hi pot haver diverses estratègies que produeixin el mateix resultat. Trobar el millor algoritme no sempre és fàcil i, de vegades, coses que poden semblar complicades, es basen en idees extremadament senzilles. Per exemple, no deixar rastre.

Veurem dues funcions ben senzilles: uns enllaços que no permeten tornar a la pàgina que els ha fet, i unes adreces de correu invisibles per als robots publicitaris.

En el primer cas, fem els enllaços a través d'una funció que reemplaça l'adreça de la pàgina actual per la del vincle que s'ha obert, utilitzant el mètode **location.replace()**. La pàgina no es guarda a l'historial i no queda cap rastre de que s'ha visitat.

Si no volem rebre correu publicitari, apliquem la segona funció. Com l'anterior, utilitza el mètode "replace" i té, a més, l'adreça trossejada (mailto:elmeunom@xtec.net). No es mostra completa ni al codi, - que és el que analitzen els robots - ni a la barra d'estat. Però funciona.

```
<script language="javascript">
function noHist(url){
  location.replace(url)
}
function noSpam(){
  location.replace( 'mai' + 'lto:elmeun' + 'om@x' +'tec.es' )
}
</script>
</head>

<body ...>
<a href="javascript:noHist( 'www.xtec.net' )">Web de la Xtec</a><br>
<a href="javascript:noSpam()">Escriu-me un mail</a><br>
...
```

Atenció a la diversitat > **Informe del client**

En aquest mòdul hem desplegat el DOM, hem vist com detectar el navegador i controlar l'entorn, i com podem afinar els estils. Intentarem posar en una pàgina alguns d'aquests aprenentatges. Crearem o modificarem l'arxiu "**index.htm**" situat a la carpeta "**m4**".

Per començar, farem un full d'estils tal com s'ha explicat a la pràctica 3. Amb el Dreamweaver, posarem títol a la pàgina, per exemple, "Informe del client", i crearem una taula de cinc files i una columna. Aplicarem els estils a les cel·les.

La primera fila serà el títol de la pàgina, en JavaScript. El títol és una propietat del document, tal com s'explica a la pràctica 1.

A la segona fila hi posarem, també en JavaScript, l'adreça de l'arxiu i la data de l'última modificació i comprovarem que, quan modifiquem i guardem l'arxiu, s'actualitza.

Detectarem el navegador i, si és Internet Explorer, informarem sobre la mida de l'arxiu utilitzant la propietat **document.fileSize**, que el navegador Netscape 4.7 no reconeix.

La tercera fila contindrà tota la informació que puguem oferir sobre el navegador.

A la quarta, informarem sobre la pantalla, colors i entorn en general. Podem utilitzar com a base el primer exemple de la pràctica 4.

A l'última fila farem un enllaç a la nostra adreça de correu de manera que no la puguin detectar els robots, tal com s'ha explicat a la pràctica 4, i posarem un vincle per tornar a la nostra pàgina principal i que no es guardi a l'historial. Farem tots dos enllaços tal com s'explica a l'últim exemple de la pràctica 2, amb la tècnica de canvi de color quan passi el ratolí, sigui quin sigui l'explorador utilitzat

Programar com cal

És important que utilitzem variables globals i funcions a la capçalera de la pàgina, de manera que, des del cos, es puguin "cridar" amb una única línia.

Tot i que en aquest exercici no és imprescindible, va bé que ens hi anem acostumant, perquè és la tècnica habitual de programar i és la que utilitzarem durant la resta del curs..

Envieu un missatge a la vostra tutoria un cop l'exercici estigui penjat a Internet. Indiqueu sempre l'adreça de forma que s'hi pugui accedir directament.

El temps al meu costat

Qui no vol tenir el temps de part seva? "Time is on my side", que deien els Stones. Si, en el mòdul anterior, hem vist com controlar el navegador i l'entorn - i actuar en conseqüència -, en aquest farem el mateix amb el temps. Ara que ja sabem com ens visiten, mirarem de saber, també, quan ho fan. I més.

La màquina té rellotge

El control del temps és important. JavaScript, com a llenguatge que s'executa en un navegador, no pot saber la data i hora del servidor, però sí les del client. Això fa que l'ús de funcions basades en aquestes dades tingui un cert risc, perquè no podem estar segurs que seran correctes. En canvi, no té importància si, per exemple, volem calcular intervals.

JavaScript proporciona l'objecte Date, que permet manejar la data i l'hora del sistema. Els mètodes d'aquest objecte ja existeixen, de manera que només els hem d'agafar i usar.

A partir d'aquest objecte podem extreure informació detallada: hores, minuts, segons, dia del mes i de la setmana, mes, any, ... i amb els mètodes podem fer operacions amb aquestes dades. En aquest mòdul aprendrem a fer rellotges i calendaris per posar a les nostres pàgines, un servei més per als visitants de la nostra web.

Programar accions

Un altre aspecte fonamental del control sobre el temps ens l'ofereixen els temporitzadors, sobretot quan es tracta d'animació. Un temporitzador no és més que un event provocat pel rellotge intern de la màquina. JavaScript els maneja de dues maneres: de forma puntual o cíclica. Amb la primera podrem programar una acció al cap d'un temps i, amb la segona, fer una acció cada interval de temps.

També mirarem com es fa una barra de progrés. De moment, aquest serà l'exercici més inútil del curs, ja que no farem progressar res més que els nostres coneixements. Ens serviran per indicar que la pàgina no s'ha mort - que continua activa, vaja - quan s'hagin de fer operacions que tinguin una bona estona la màquina o la línia ocupada.

Finalment, veurem un dels usos més interessants dels temporitzadors: les accions en segon pla, que ens han de permetre aprofitar el temps d'inactivitat per avançar continguts. Es tracta de treure el màxim partit de les connexions quan sabem què ha de venir a continuació. A la pràctica 4 en veurem un exemple complet, una mini-exposició virtual de paisatges pintats per Van Gogh, tot un luxe!

Amb tot això, ... algú sap quina hora és?

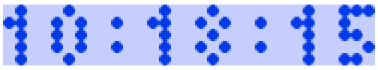
10:17:59

En aquest mòdul

Pràctica 1: **Rellotges**
Pràctica 2: **Calendaris**
Pràctica 3: **Barra de progrés**
Pràctica 4: **Accions en segon pla**

Exercicis: **Temps controlat**

Quan escrivim codi JavaScript, cal recordar que Dreamwever disposa d'una eina, el depurador, que ens facilita la detecció i el tractament d'errades. Es pot activar des del menú "Archivo - Depurar en navegador" o fent clic al botó de la barra d'estat "Vista previa/depurar en explorador".



En aquesta pràctica es veurà com podem mesurar i mostrar el temps en les nostres aplicacions web. Introduirem inicialment els conceptes bàsics que ens han de permetre utilitzar el rellotge del sistema i les formes més habituals de mostrar-lo en la pàgina web.

També veurem com podem implementar un cronòmetre que ens permeti mesurar intervals de temps acotats entre dos events, que poden ser originats per l'usuari o per altres events de la web.

Finalment es mostrarà un sistema de presentació més elaborat: un rellotge digital amb presentació mitjançant una matriu de punts.

Doncs bé, comencem?

Elements del llenguatge en aquest capítol.

Mètode **setTimeout()** : Inicia un temporitzador.
Mètode **clearTimeout()** : Atura un temporitzador que estigui actiu.
Propietat **status** : Propietat de l'objecte window que conté el contingut que es veu a la barra d'estat.
Mètode **split()** : Descompon una cadena de text, segons l'element separador, creant un array dels elements separats.
Objecte **Date** : Ens permet treballar amb dates i hores.
Mètodes diversos de l'objecte Date.

1.- L'hora actual.

La mesura del temps és una utilitat molt habitual en els llenguatges de programació i JavaScript no podia ser menys. El llenguatge ens proporciona un objecte d'aplicació específica per aquest concepte. És l'objecte **Date**.

L'objecte **Date** ens proporciona un conjunt de mètodes i propietats per treballar amb dates i hores. Qualsevol variable instanciada de l'objecte *Date* contindrà internament el nombre de mil·lisegons (positius o negatius) que han passat des del dia 1 de Gener de 1970.
per crear una instància de l'objecte *Date*, s'utilitza el constructor **new** de la forma següent:

```
var data = new Date(paràmetres);
```

Els *paràmetres* poden ser:
- El nombre de mil·lisegons transcorreguts des de l' 1 de Gener de 1.970.
- Una cadena de text, en el format data i/o hora reconegut per l'objecte *Date*.
- Un conjunt de valors numèrics, separats amb coma que representen a: any, mes, dia, hora, minut i segon. Els tres últims són opcionals.
- Cap paràmetre.

Si no s'indica cap paràmetre, l'objecte *Date* agafarà la data i hora de l'ordinador que s'estigui utilitzant.

Un cop s'hagi creat una instància de l'objecte *Date* (en l'exemple anterior, la variable **data**) podrem aplicar sobre aquesta variable un conjunt de mètodes que ens permeten obtenir (**get...**) i assignar (**set...**) diferents valors que composen aquesta data.

Entre els més importants tenim:
getHours().-Retorna l'hora.
setHours().- Assigna l'hora.
getMinutes().- Retorna el nombre de minuts.
setMinutes().- Assigna el nombre de minuts.
getSeconds().- Retorna el nombre de segons.
setSeconds().- Assigna el nombre de segons.

getFullYear().- Retorna l'any indicat amb quatre xifres.
setFullYear().- Assigna l'any indicat amb quatre xifres.
getMonth().- Retorna el número del mes. 0- Gener ... 11- Desembre
setMonth().- Assigna el número del mes. 0- Gener ... 11- Desembre.
getDate().- Retorna el número del dia.
setDate().- Assigna el número del dia.
getDay().- Retorna el número del dia de la setmana. 0- Diumenge ... 6- Dissabte.

El següent codi ens mostra uns exemples d'ús d'alguns d'aquests mètodes.

La funció **horaAra()** és la responsable de detectar la data i hora actual del rellotge del sistema, amb la línia var ara = new Date(), es declara una variable instanciada de l'objecte *Date* que conté la data i hora actual, ja que no s'ha passat cap paràmetre al constructor *Date()*.

Un cop es té aquest valor, declarem i assignem la variable hora. El valor assignat és el que obtenim del mètode **getHours()** aplicat a la variable **ara**. Però abans fem una comprovació per si de cas el seu valor fos inferior a 10. Per fer-ho utilitzem l'operador condicional ternari **?:**, tal com s'indica a la següent línia:

```
(ara.getHours()<10) ? "0"+ara.getHours() : ara.getHours();
```

La condició és: **(ara.getHours()<10)**. Si aquesta és certa, és a dir el valor de l'hora solament té un dígit, retornem un **"0"** afegit al valor retornat per **ara.getHours()**. De cas contrari, retornem el valor subministrat per **ara.getHours()** tal qual. Per tant, el contingut de la variable *hora* sempre contindrà dos dígits, un d'ells serà un zero a l'esquerra si és necessari.

Per a les variables *minuts* i *segons* es fa un procés similar a l'anterior.

Un cop es tenen els tres valors de les variables corresponents és necessari mostrar-los dins la pàgina web. La forma d'implementar aquesta operació pot ser molt diversa, i de fet, per les pàgines d'Internet es poden trobar moltes formes de mostrar la data i hora, algunes d'elles amb efectes vistosos i espectaculars.

En aquest primer exemple serem una mica més modestos i ens limitarem a mostrar-ho sobre dos elements molt simples d'implementar: un camp de formulari i la barra d'estat del navegador.

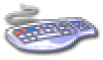
La línia document.forms.form1.contenidor.value = hora + ":" + minuts + ":" + segons; s'encarrega de mostrar el valor de hora, minuts i segons amb els **:** enmig dins un camp de text de nom **contenidor** situat dins un formulari de nom **form1** situat en el bloc <*body*> de la pàgina.

La línia window.status = hora + ":" + minuts + ":" + segons; també representa el valor de hora, minuts i segons amb uns **:** entre mig, a la barra d'estat del navegador.

status és una propietat de l'objecte **window** que conté el valor que es mostra a la barra d'estat del navegador. Observeu que és molt senzill utilitzar aquesta sortida d'informació. De fet, aquesta propietat és molt utilitzada per fer-hi efectes de text. Ha Internet es pot veure moltes aplicacions que utilitzen aquesta propietat "status" per fer efectes més o menys elaborats amb cadenes de text. A més, és totalment compatible amb qualsevol tipus de navegador.

Per acabar la descripció de la funció ens queda la línia setTimeout("horaAra()",1000); Aquesta s'encarrega d'engegar un temporitzador que al cap de 1000 mil·lisegons torni a executar de nou la funció **horaAra()**. D'aquesta forma s'aconsegueix que els rellotges s'actualitzin cada segon. Proveu d'eliminar aquesta línia i observareu com el valor del rellotge no s'actualitza a no ser que es recarregui la pàgina.

La funció *horaAra()* s'invoca per primer cop mitjançant l'event **onLoad** situat a la directiva <body>.



```
<html>
<head>
<title>Rellotges</title>
<script language="JavaScript">
```

```
function horaAra()
{
  var ara = new Date();
  var hora = (ara.getHours()<10) ? "0"+ara.getHours() : ara.getHours();
  var minuts = (ara.getMinutes()<10)?"0"+ara.getMinutes():ara.getMinutes();
  var segons = (ara.getSeconds()<10)?"0"+ara.getSeconds():ara.getSeconds();
  document.forms.form1.contenidor.value = hora + ":" + minuts + ":" + segons;
  window.status = hora + ":" + minuts + ":" + segons;
  setTimeout("horaAra()",1000);
}
</script>
</head>

<body bgcolor="#FFFFE7" onLoad="horaAra();">
<p><p><p>
<form name="form1">
<center>
  Hora actual:   <input type="text" name="contenidor" value="">
</center>
</form>
<p><p>
També ho pots veure a la barra d'estat:
</body>
</html>
```

2.- La data actual

Per obtenir la data actual indicada pel sistema es pot fer utilitzant un codi estructurat de forma similar a l'exemple anterior. Però, en aquest cas, s'ha de tenir en compte que **getMonth()** retorna el número de mes entre 0 i 11, i **getDay()** retorna el número de la setmana entre 0 i 6 (diumenge - dissabte). Per tant s'ha de fer la conversió.

És pot observar com s'assigna a una cadena la llista de noms de mesos i la llista de noms de dies de la setmana. Posteriorment amb el mètode **split()** es crea un array a **nomMes** i a **nomDia**, utilitzant com a separador l'espai (" "). Ara *nomMes* i *nomDia* són dos arrays, on el primer element serà el primer de la llista i l'últim element serà l'últim de la llista.
per referenciar cada element de l'array es podrà fer amb *nomMes[index]* i *nomDia[índex]*, on *índex* és el valor que ens retorna **getMonth()** i **getDay()** respectivament.

Finalment, mitjançant **document.write**, s'escriu la cadena de text sobre el document per formar la data actual.

```
  ......      Codi dins el bloc <body>

<script language = "JavaScript">
  var ara=new Date();
  var nomMes="Gener Febrer Març Abril Maig Juny Juliol Agost Setembre Octubre
                                                   Novembre Desembre";
  var nomDia="Diumenge Dilluns Dimarts Dimecres Dijous Divendres Dissabte";
  nomMes= nomMes.split(" ");
  nomDia= nomDia.split(" ");
  document.write(nomDia[ara.getDay()] + ", a " + ara.getDate() + " de " +
                              nomMes[ara.getMonth()] + " de " + ara.getFullYear() );
</script>

  ......
```

3.- Cronòmetre

La implementació de cronòmetres consisteix en determinar un valor de Date inicial i un valor de Date al final de l'interval. La diferència serà el temps transcorregut.

Recordem que una variable instanciada de l'objecte Date conté el nombre de mil·lisegons des de l'1 de Gener de 1.970, per tant, la diferència entre dues variables tipus Date donarà com a resultat els mil·lisegons transcorreguts.

El següent exemple es basa en aquest concepte. Quan fem un clic a sobre el botó "Inicia", es crea una instància de Date en la variable **abans**. Aquesta variable conté el temps inicial. Posteriorment, dins el mateix controlador d'event **onClick**, s'invoca a la funció **cron()**.

La funció **cron()** torna a instanciar una nova variable tipus Date de nom **ara**, i assigna a la propietat **value** del camp de text *crono* del formulari *form1*, la diferència respecte **abans** dividit per 1.000 per obtenir el resultat en segons.

També s'efectua la mateixa operació per assignar-ho a la propietat **status** de l'objecte **window**. És a dir, també es veurà el comptador a la **barra d'estat** de la finestra.

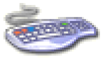
En acabar de fer les dues assignacions per mostrar el resultat de la diferència, s'engega un temporitzador mitjançant **setTimeout()**. La funció d'aquest temporitzador és tornar a invocar la funció **cron()** al cap de 10 mil·lisegons. A la vegada, **setTimeout()** retorna un valor a **tID**, el valor retornat és l'identificador numèric del temporitzador que està en execució.

Quan desitgem aturar el cronòmetre, farem clic a sobre del botó "Atura", el seu event onClick invocarà a la funció **clearTimeout()** passant-li el paràmetre corresponent a l'identificador del temporitzador que estigui actiu en aquest moment. La funció **clearTimeout()** atura el temporitzador indicat en el seu paràmetre **tID**. Amb això aconseguim aturar el cronòmetre.

Observeu com a la variable *abans* solament se li assigna un valor al principi del procés, quan es fa clic a sobre de "Inicia". En canvi, a la variable *ara* se li assigna un valor cada cop que s'invoca a la funció **cron()** (cada 10 ms.). Lògicament, el temps transcorregut (*ara - abans*) cada cop serà més gran.

Una aplicació similar a la que s'ha descrit en aquest apartat consisteix a mesurar el temps que un visitant ha estat visualitzant una pàgina web. Quan tanqueu aquesta pàgina web, us apareixerà un missatge *alert()* que us mostrarà el temps que heu estat visualitzant la pàgina.

L'efecte s'ha aconseguit de forma similar a la descrita pel cronòmetre, desant en una variable el temps actual mitjançant l'event *onLoad* situat en la directiva *<body>*. En tancar la pàgina, mitjançant un altre event *onUnload()* situat a la directiva *<body>* es torna a determinar el temps actual i es mostra mitjançant l' *alert()* la diferència de temps entre les dues variables.



```
<html>
<head>
<title>Cronòmetre</title>
<script language="JavaScript">
  var abans;
  var tID;
  function cron()
  {
    var ara=new Date();
    document.forms.form1.crono.value=(ara-abans)/1000 + " s.";
    window.status=(ara-abans)/1000 + " s.";
    tID=setTimeout("cron()",10);
  }
</script>
</head>

<body bgcolor="#FFFFE7">
  <form name="form1">
  <input type="text" name="crono" value=""> 
  <input type="button" value="Inicia" onClick="abans=new Date(); cron();"> 
  <input type="button" value="Atura" onClick="clearTimeout(tID);">
  </form>
</body>
</html>
```

4.- Millorem la presentació de sortida.

Fins ara ens hem limitat a mostrar la sortida del rellotge, data i cronòmetre en tres elements relativament fàcils d'implementar com són un camp de text de formulari, la barra d'estat del navegador i directament sobre el bloc 'body' del document amb *document.write()*.

Si volem millorar l'aspecte gràfic de sortida de les dades es pot fer mitjançant la utilització d'estils, però, en aquest apartat es proposa la creació de dígits numèrics mitjançant una

matriu de punts.

La matriu de punts de cada dígit és de 3 columnes per 5 files. En cada matriu assenyalem amb un " * " el punt que ha de quedar activat a ON i amb un " · " el punt que ha de quedar a OFF. Tot el conjunt de dígits numèrics i símbol " : " format amb matrius de punts s'introdueix en una variable de cadena de nom **caràcters** .

En la secció de codi es pot veure la variable "caràcters". S'ha indicat mitjançant línies numerades per veure millor com estan formats els caràcters, però, en la implementació dins el codi, la variable ha de tenir tots els punts i asteriscs indicats en una sola línia, sense cap espai, de la forma següent:

var caracters = ".*.*.**.**.*.*..*.**..*. .... ";

Les variables **col** i **fil** han de contenir el nombre de columnes i files de la matriu, respectivament.

La funció **digital()** rep com a paràmetre una cadena de text, de longitud indeterminada, formada per caràcters dels que estan definits dins de la variable "caràcters". Seguidament, analitza cadascun dels caràcters de la cadena de text i extreu la seva representació segons la matriu de punts.

En funció de quina sigui la representació del caràcter, assignarà una imatge de nom "on.gif" si és un asterisc o bé una imatge de nom "off.gif" si és un punt. D'això se n'encarrega la funció auxiliar **ferImg()**.

En aquest exemple s'ha utilitzat la imatge per a "on.gif" i la imatge per a "off.gif". Aquestes imatges es poden modificar pel que fa a la mida i colors per obtenir altres resultats gràfics.

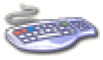
La funció "ferImg()" rep com a paràmetres cad, f i ncar. **cad** és un array format per tots els caràcters de la cadena "caràcters". **f** és el número de fila que s'està formant en aquest moment i **ncar** és la posició del caràcter dins de l'array (el 0 ocupa la posició 0, el 1 la posició 1, el : la posició 10 .... , teniu en compte que una posició d'un caràcter dins de l'array ocupa (fil*col) elements de l'array.

Finalment, la funció **digital() retorna una taula** que conté les referències HTML per mostrar les imatges "on.gif" i "off.gif", ordenades adequadament per formar el text desitjat. Aquesta taula es pot situar on es desitgi dins la pàgina web.

Aquesta funció és molt versàtil i permet altres modificacions. Noteu que no estem pas restringits a utilitzar solament dígits numèrics. Podem crear tot l'alfabet amb majúscules, minúscules i caràcters especials i afegir-los a la variable "caràcters" i a la vegada, també s'hauran d'afegir els nous caràcters dins l'estructura "case", de forma similar a com s'ha fet entre les línies 10 i 20 de la funció **digital()**.

Per millorar l'aspecte dels dígits i donar-los més "resolució" es pode crear matrius de punts amb més files i columnes (p.e. 7x5, 9x5, etc.) i incorporar-los a la variable "caràcters". En aquest cas s'hauran de modificar també les variables "fil" i "col".

Modificant les imatges "on.gif" i "off.gif", pel que fa a colors i mida, es poden aconseguir altres variacions d'efecte estètic.



```
1  var caracters="
2  . * .
3  * . *
4  * . *
5  * . *
6  . * .

7  . * .
8  * * .
9  . * .
10 . * .
11 . * .
12 . * .
13 * . *
14 . * .
15 * . .
16 * * *
17 * * .
18 . . *
19 . * .
20 . . *
21 * * .
22 * . .
23 * . *
24 * * *
25 . . *
26 . . *
27 * * *
28 * . .
29 * * .
30 . . *
31 * * .
32 . * .
33 * . .
34 * * .
35 * . *
36 . * .
37 * * *
38 . . *
39 . * .
40 . * .
41 . * .
42 . * .
43 * . *
44 . * .
45 * . *
46 . * .
47 . * .
48 * . *
49 . * *
50 . . *
51 . * .
52 . . .
53 . * .
54 . . .
55 . * .
56 . . .
57 ";
```

```
var col = 3;
var fil = 5;
```

```
1  function digital(text)
```

```
{
  var car;
  var cad = caracters.split("");
  var taula="<table border='0' cellspacing='0' cellpadding='0'>\n";
  for (var f=0; f<fil; ++f) {
    taula += "<tr><td>"
    for (car=0; car<text.length; ++car) {
      switch (text.substring(car,car+1)) {
         case '0': taula += ferImg(cad,f,0);break;
         case '1': taula += ferImg(cad,f,1);break;
         case '2': taula += ferImg(cad,f,2);break;
         case '3': taula += ferImg(cad,f,3);break;
         case '4': taula += ferImg(cad,f,4);break;
         case '5': taula += ferImg(cad,f,5);break;
         case '6': taula += ferImg(cad,f,6);break;
         case '7': taula += ferImg(cad,f,7);break;
         case '8': taula += ferImg(cad,f,8);break;
         case '9': taula += ferImg(cad,f,9);break;
         case ':' : taula += ferImg(cad,f,10);break;
       }
       if (car != text.length-1) taula += "<img src='off.gif'>";
     }
     taula += "</td></tr>\n";
   }
   taula += "</table>";
   return taula;
 }
```

```
function ferImg(cad,f,ncar)
{
  var img = "";
  for (var i=0; i<col; ++i) {
    img += "<img src='";
    img += (cad[col*fil*ncar+col*f+i] == '*')?"on.gif":"off.gif";
    img += "'>";
  }
  return img;
 }
```

```
<html>
<head>
<title>Rellotge digital</title>
<script language="JavaScript">
  ...... introduïm la variable "caracters" .....
  ...... introduïm les variables "col" i "fil" .....
  ...... introduïm la funció digital() .....
  ...... introduïm la funció ferImg() .....
</script>
</head>
<body>
<script language="JavaScript">
  var ara = new Date();
  var hora = (ara.getHours()<10)?"0"+ara.getHours():ara.getHours();
  var minuts = (ara.getMinutes()<10)?"0"+ara.getMinutes():ara.getMinutes();
  var segons = (ara.getSeconds()<10)?"0"+ara.getSeconds():ara.getSeconds();

  document.write( digital(hora + ":" + minuts + ":" + segons) );
</script>
</body>
</html>
```

Hem vist en la pràctica anterior com podem determinar dades referents a la data i hora del sistema i com les visualitzem en el document. Ara es tracta de veure una aplicació una mica més complexa: La creació d'un calendari.

Juliol 2002						
Dl	Dm	Dc	Dj	Dv	Ds	Dg
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Gairebé tota la pràctica girarà a l'entorn d'una funció encarregada de crear el calendari. Aquesta serà la funció més complexa i a la que dedicarem més temps a explicar.

Un cop tindrem feta la funció de creació del calendari, la complementarem amb exemples d'utilització i altres funcionalitats addicionals per fer-la més atractiva i interactiva.

Conceptes JavaScript en aquest capítol.

Objecte **Date** : Proporciona mètodes per treballar amb dates.
Mètode **getFullYear()** : Retorna el número de l'any que conté l'objecte Date.
Mètode **getMonth()** : Retorna el número del mes que conté l'objecte Date.
Mètode **getDay()** : Retorna el número del dia de la setmana que conté l'objecte Date.
Mètode **getDate()** : Retorna el dia que conté l'objecte Date.
Mètode **setDate()** : Estableix el número del dia que conté l'objecte Date.
Propietat **opener** : Propietat de l'objecte Window que conté el nom de la finestra mare.

1.- Descripció del codi. Inicialitzacions.

Començarem dissenyant el codi per crear un calendari mitjançant una funció. S'ha elegit fer-ho en forma de funció ja que així serà més fàcilment transportable.

La funció de nom **calendari** haurà de rebre dos paràmetres:, l'any i el número de mes (0 .. 12) que desitgem visualitzar. La seva sintaxi serà la següent:

```
calendari(pAny, pMes);
```

Si els paràmetres pAny i/o pMes valen 0, la funció prendrà per defecte l'any i el més del sistema. Un cop invocada la funció, aquesta **retorna una cadena** que conté tot el codi HTML de la taula que formarà el calendari.

De forma genèrica, la funció està dividida en dues parts principals: la inicialització i la creació de la taula dins una variable de cadena de nom **cal**. Cada part es pot subdividir en:

Inicialitzacions prèvies:
- Inicialitzar Arrays i variables.
- Obtenir la data actual del sistema.
- Si els paràmetres que se li han passat són diferents de zero, substituirà la data d'avui per la dels paràmetres.
- Determina si es tracta d'un any de traspàs.
- Determinar el dia de la setmana del primer dia del mes.

Creació de la taula calendari:
- Representar la primera fila de la capçalera on hi ha el nom del mes i l'any.
- Representar la fila on hi ha els noms dels dies de la setmana.
- Representar les primeres cel·les en blanc, si n'hi ha.
- Representar les cel·les de les files on hi ha els números dels dies.
- Representar les últimes cel·les en blanc, si n'hi ha.

El codi i la descripció de la funció és extensa i per tant es dividirà en blocs funcionals, al final de la pràctica, s'ha d'introduir tot el codi en conjunt, per treballar dins una pàgina HTML d'aplicació. Per facilitar la lectura del codi, les línies de la funció *calendari()* estan

numerades. En l'aplicació pràctica, els números de línia no s'han de posar.

La funció és la següent:

```
1 function calendari(pAny, pMes)
2 {
3   var nomDiaSem = new Array("Dl", "Dm", "Dc", "Dj", "Dv", "Ds", "Dg");
4   var nomMes = new Array("Gener", "Febrer", "Març", "Abril", "Maig", "Juny", "Juliol",
                        "Agost", "Setembre", "Octubre", "Novembre", "Desembre");
5   var diesMes = new Array(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
6   var ampleCol = 30;
```

En aquestes primeres línies tenim la capçalera de la funció calendari. Se li passen dos paràmetres, **pAny** i **pMes**. Les línies 3, 4 i 6 són variables amb continguts que es poden modificar per canviar les abreviacions, l'idioma, l'ample de columna, etc.

La línia 5 conté els dies de cada mes començant pel Gener, cal observar que en la dada que correspon al més de Febrer s'hi ha posat 28 dies. Si l'any és de traspàs, aquest valor s'haurà de canviar a 29. Més endavant es fa aquesta comprovació i es modifica si s'escau.

La variable **ampleCol** conte l'amplada de cada cel·la de la taula del calendari en píxels, si es desitja fer el calendari més ample es pot fer canviant aquest valor.

```
7    var avui = new Date();
8    if (pAny!=0) var any=pAny;
9    else var any=avui.getFullYear();
10   if (pMes!=0) var mes=pMes-1;
11   else var mes=avui.getMonth();
12   var dia=avui.getDate();
13   var dataCal = new Date(any, mes, dia);
14   if ( ((any%4==0) && (any%100 != 0)) || (any%400==0) ) diesMes[1] = 29;
15   var numDies = diesMes[mes];
```

Cada cop que s'utilitza una variable per primera vegada ha d'anar precedida per la corresponent declaració de variable **var**.

Mitjançant *Date()*, a la línia 7, s'obté la data del sistema i s'assigna a la variable tipus Date de nom **avui**. Aquesta data s'utilitzarà per assignar-la a les variables **any**, **mes** i **dia**, a no ser que s'hagi posat en els paràmetres *pAny* i/o *pMes* un valor diferent de 0.

Les línies 8 i 9 s'encarreguen d'assignar a la variable *any*, l'any que indica el sistema, en cas que *pAny* valgui 0 o en cas contrari, l'any indicat per *pAny*. Les línies 10 i 11 fan el mateix per la variable *mes*. La variable *dia* sempre contindrà el valor de la data actual (línia 12).

Cal notar que per determinar el valor del dia, mes i any actuals, s'utilitzen els mètodes **getDate()**, **getMonth()** i **getFullYear()** de Date. A la línia 10, es resta una unitat de *pMes* per que el paràmetre *pMes* enumera els mesos entre 1 i 12, en canvi la variable mes els enumera entre 0 i 11.

A la línia 13 es crea un nou objecte tipus Date, però en aquest cas, ja no és amb la data per defecte del sistema, sinó amb la data que nosaltres li passem com a paràmetres. Així doncs, **dataCal** contindrà la data que es mostrarà en el calendari (ens interessa per tenir el *mes* i *any* que s'ha de mostrar). Cal recordar que *avui* sempre conté la data actual del sistema i no té per que ser la mateixa que **dataCal**. Si totes dues són diferents serà perquè algun dels paràmetres passats a la funció és diferent de 0.

La línia 14 verifica si l'any és de **traspàs** o no, per això verifiquem si *any* és un múltiple de 4 i no és múltiple de 100, o bé, *any* és múltiple de 400. En cas que la condició sigui certa, voldrà dir que és un any de traspàs i per tant, el nombre de dies del mes de Febrer haurà de valer 29. Amb **diesMes[1]**, fem referència al segon element (Febrer) de l'array *diesMes* i el modifiquem, si s'escau.

Finalment, assignem a la variable **numDies** el nombre de dies que tindrà el mes a representar, obtenint el valor de l'Array *diesMes*.

```
16 var primerDia = dataCal; //Determinem el dia de la setmana del primer dia del mes
17 primerDia.setDate(1);
18 var diaSem1 = (primerDia.getDay()==0)?7:primerDia.getDay(); //1->dilluns,
                                                              7->diumenge
```

Una altra dada que és necessari conèixer és el dia de la setmana en què comença el mes, és a dir, quin dia de la setmana és el dia 1 del més a representar.

Per això ens pot ser molt útil el mètode **getDay()** de Date. Aquest mètode retorna un valor enter entre 0 i 6, on 0 correspon al Diumenge i 6 al Dissabte. Noteu que l'ordenació dels dies és la pròpia dels països anglosaxons.

Per determinar aquest dia, assignem, en la línia 16, a **primerDia** el valor de *dataCal*, es a dir, ara *primerDia* també serà un objecte tipus Date i conté la mateixa data que *dataCal*. A la línia 17, utilitzem el mètode **setDate()**, aquest mètode assigna a un objecte tipus Date el valor numèric corresponent al dia passat com a paràmetre. Per tant ara *primerDia* conté la data del primer dia del mes sense haver modificat el valor del mes i de l'any.

Finalment, a la línia 18, utilitzant el mètode **getDay()**, assignem el número del dia de la setmana a *diaSem1*. Com que nosaltres volem tenir una referència dels dies de la setmana segons els nostres usos i costums i no, tal com ho fan en els països anglosaxons, hem d'efectuar una conversió amb l'ajut de l'operador ternari condicional **? :** .

Aquest operador ternari **? :** ja s'ha estudiat al mòdul 2, pràctica 4 i en el mòdul 3 pràctica 3. En aquest cas, té la funció de seleccionar entre dos valors en funció de la condició. Si el valor retornat per **getDay()** és 0, s'assignarà un 7 a **diaSem1**. En canvi, per a qualsevol altre valor retornat per *getDay()*, serà assignat directament a *diaSem1*. Amb això tindrem que *diaSem1*, valdrà un valor entre 1 i 7 on 1 correspondrà a Dilluns i 7 a Diumenge.

Fins aquí ja s'han inicialitzat tots els valors previs a la creació del calendari. I en resum, sabem que les variables més importants que necessitarem a partir d'ara són:

avui	Objecte Date amb la data actual del sistema (pot coincidir, o no amb dataCal).
dataCal	Objecte Date amb la data que es vol representar al calendari.
dia	Valor numèric corresponent al dia d'avui.
mes	Valor numèric corresponent al mes a representar. (0 - Gener, 11 - Desembre)
any	Valor numèric corresponent a l'any a representar.
diaSem1	Valor numèric del dia de la setmana del dia 1 (1 - Dilluns, 7 - Diumenge).
ampleCol	Amplada en píxels de cada columna del calendari.
nomMes	Array amb els noms dels mesos.
nomDiaSem	Array amb els noms dels dies de la setmana.

2.- Descripció del codi. Creació de la taula - calendari.

Les línies de codi que venen a partir d'ara s'encarreguen de crear la taula - calendari. Únicament s'indica el codi imprescindible per a la correcta visualització, més endavant ja s'afegiran altres elements estètics i de funcionalitats més específiques.

Gairebé tota la representació es basa en l'escriptura de codi HTML, formant l'estructura de la taula. Tota aquesta estructura de la taula s'introduirà seqüencialment en una variable de tipus cadena de nom **cal**. Al final, tot el contingut d'aquesta variable serà retornat per la funció.

Les cel·les que hagin d'estar en blanc, s'omplen amb espais ** **. Totes les línies de codi són per crear una estructura de taula similar a la següent, on, evidentment, els valors canviaran segons quin sigui el mes a representar.

Gener 2003						
Dl	Dm	Dc	Dj	Dv	Ds	Dg
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

```
19  var cal="<table border='1'>\n";
20  cal += "<tr><th colspan='7'>" + nomMes[mes] + " " + any
                                              + "</th></tr>\n";
```

La línia 19 inicialitza la taula. Declarem inicialment la variable **cal** i li assignem la directiva <table...>.
La línia 20 s'encarrega d'escriure la capçalera de la taula, resultant un format similar al següent:

Gener 2003

Noteu com en la línia 20 s'utilitza l'operador **+=**, en aquesta aplicació, aquest operador assigna a la variable de l'esquerra la concatenació dels valors que hi ha a la dreta i a l'esquerra. Per exemple: Si 'vble' és una variable que val "cadena1" i fem vble += " cadena2", la variable 'vble' valdrà "cadena1 cadena2", això és equivalent a fer: vble = vble + "cadena2".

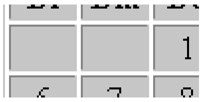
```
21  cal += "<tr>";
22  for(var i=0; i<7; i++) cal += "<th width=' " + ampleCol + " '>" +
                                              nomDiaSem[i] + "</th>";
23  cal += "</tr>\n";
```

Aquest bloc de codi recorre l'array **nomDiaSem** i crea una fila de la taula amb 7 cel·les, una per a cada dia. En cada cel·la s'introdueix el nom del dia de la setmana llegit de l'array **nomDiaSem**. En aquesta secció també s'especifica l'amplada de la cel·la segons el valor d' **ampleCol**, per tant, s'està introduint l'ample de les columnes de tota la taula. El tros de taula creat serà el següent:

Dl	Dm	Dc	Dj	Dv	Ds	Dg

```
24  var columna = 1;
25  cal += "<tr>";
26  for(i=1; i<diaSem1; i++) {
27     cal += "<td> </td>";
28     ++ columna;
29  }
```

Segons quin sigui el mes a representar, pot passar que les primeres cel·les estiguin en blanc. El blucle **for**, entre les línies 26 i 29, s'encarrega d'escriure cel·les en blanc fins a arribar al valor de **diaSem1**, recordem que aquesta variable té el valor numèric del nom del dia corresponent al dia 1r del mes.



A la línia 24 s'inicialitza una variable de nom **columna**, aquesta variable s'utilitzarà en aquest bloc i els següents per saber en quina columna s'està escrivint de les 7 que té el calendari. Això servirà per saber quan s'ha de fer un canvi de fila.

```
30  for(i=1; i<=numDies; i++) {
31     cal += "<td><center>" + i + "</center></td>";
32     if(++columna > 7) {
33        cal += "</tr>\n";
34        if (i < numDies) cal += "<tr>";
```

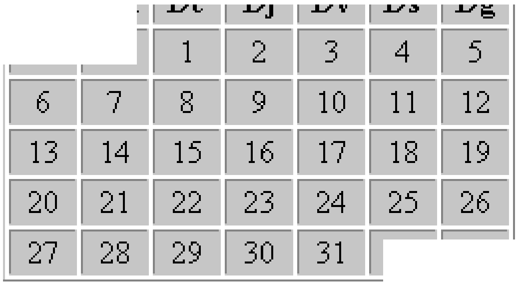
```
35          columna = 1;
36      }
37   }
```

Aquestes línies són les responsables d'escriure el cos principal del calendari. Per això es crea un bucle **for** on la variable **i** canviarà des de 1 fins al màxim nombre de dies del calendari (**numDies**).

A la línia 31, per cada valor d'**i,** s'afegeix a **cal** una cel·la que contindrà el numero de dia corresponent que s'obté de la variable **i**.

A més, a cada cel·la, també s'incrementa el valor de la variable **columna**, si aquesta valgués 7, voldria dir que ja s'ha completat una fila i s'ha de començar la següent, per tant, si *columna* val 7, s'afegeix a **cal** un </tr> per tancar la fila. En el cas que no s'hagi acabat d'escriure el calendari també s'afegeix un <tr> per obrir la nova fila i es torna a posar el valor de *columna* a 1, tal com es pot veure a les línies 33, 34 i 35.

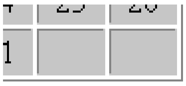
Al sortir del bucle **for**, ja s'hauran afegit a *cal* totes les cel·les que tenen un contingut numèric corresponent a un dia del mes, de forma similar a la següent imatge:



		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

```
38   if (columna!=1) {
39      for(i=0; i <= 7-columna; ++i) cal += "<td> </td>";
40      cal += "</tr>\n";
41   }
42   cal += "</table>";
43
44   return cal;
45 }
```

Finalment, en el cas que el valor de **columna** sigui diferent de 1, voldrà dir que encara no s'ha acabat d'omplir amb cel·les l'última fila. Per això, s'inicia un altre bucle **for** que introdueix les cel·les en blanc necessàries per completar la fila. Si *columna* valgués 1, (línia 38) no faria falta acabar de completar la fila, ja que aquesta hauria acabat amb un valor numèric corresponent a un número de dia.



La línia 42 tancarà la taula. Amb tot això ja estarà creat el calendari i també haurà acabat la funció. Ara solament queda **retornar** el valor de la variable **cal,** que conté tota la informació HTML de la taula creada representant el calendari desitjat, tal com es fa a la línia 44.

Per provar el seu funcionament s'ha de crear una pàgina web que incorpori aquesta funció i l'executi. El següent exemple ens mostra el seu funcionament. En aquest cas s'invoca a la funció amb els paràmetres 0 per l'any i 0 pel mes, per tant, ens mostrarà el calendari del mes i any que indiqui el rellotge del sistema. Si volem un altre mes i/o any s'haurà de canviar el valor dels paràmetres de la invocació de la funció dins el bloc <body>.

```
<html>
<head>
<title>Calendari</title>
<script language="JavaScript">
   ...codi corresponent a la funció calendari()... descrit des de la línia 1 fins la 45
</script>
</head>
<body>
<script language="JavaScript">
```

```
    document.writeln(calendari(0,0));
</script>
</body>
</html>
```

3.- Afegim colors i altres funcionalitats.

El que ve a continuació són variacions sobre el codi anterior de la funció **calendari()**. No es rescriurà tot el codi de nou, solament s'indicaran les línies que es modifiquen, s'esborren o s'afegeixen per introduir les noves funcionalitats.

Si s'afegeix alguna línia al codi de la funció original, aquesta línia portarà com a número el mateix de l'anterior seguit d'una lletra (p. e. 32, 32a, 32b, ...).

Es tracta d'**afegir colors** a les diferents cel·les del calendari, a més es desitja que la cel·la que mostra el dia, mes i any actuals quedi marcada amb un altre color. Per fer-ho s'afegeix la directiva d'HTML **bgcolor** als <th> i <td> corresponents per indicar el color de fons desitjat. Així es fa a les línies 20, 22, 27, 31 i 39. Per fer que la cel·la que indica el dia actual quedi d'un altre color es fa mitjançant un condicional **if**. Aquest condicional queda indicat en les noves línies 31a i 31b. Si el dia, any i hora coincideix amb el dia, any i hora actuals, la cel·la quedarà marcada en vermell, en cas contrari en groc.

A continuació podem veure les modificacions i afegits a fer (per provar-ho es pot fer amb la mateixa plana HTML definida al final de l'apartat anterior):

Modificar:

```
20 cal += "<tr><th colspan='7' bgcolor='#ffff00'>" + nomMes[mes]
                                                  + " " + any + "</th></tr>";
22 for(var i=0; i<7; i++) cal += "<th width=' " + ampleCol +
                                 " ' bgcolor='#ffff00'>" + nomDiaSem[i] + "</th>";
27 cal += "<td bgcolor='#ffffc0'> </td>";
39 for(i=0; i <= 7-columna; ++i) cal += "<td
                                                     bgcolor='#ffffc0'> </td>";
```

Modificar i afegir:

```
31 cal += "<td bgcolor=";
31a if (i==dia && avui.getMonth()==dataCal.getMonth() &&
        avui.getFullYear()==dataCal.getFullYear()) cal += " '#ff8080' ";
31b else cal += " '#ffffc0' ";
31c cal += "><center>" + i + "</center></td>";
```

4.- El calendari interactiu.

Des del principi, ja s'ha dit que la funció **calendari()** que hem creat admet dos paràmetres, l'any i el mes. Fins ara no els hem fet anar i ens hem limitat a posar-hi 0 per l'any i 0 pel dia. Per tant, sempre ens apareix el calendari del mes actual.

En aquest apartat, es tracta de fer que el nostre **calendari** sigui **interactiu**, és a dir, des d'una finestra mare, hem de poder elegir el mes i/o l'any desitjat i fer que es mostri el corresponent calendari en una finestra filla.

Encara més, també farem que cada dia del calendari sigui un **link** que al fer-li clic a sobre ens indiqui, en un quadre de text de la finestra mare, la data seleccionada.

Com en l'apartat anterior, tampoc rescriurem el codi de la funció *calendari()*, simplement afegirem i/o modificarem aquelles línies que facin falta.

La funció esta preparada per rebre paràmetres, però, no n'esta per fer que cada dia sigui un 'link'. Per facilitar la tasca de posteriors modificacions, crearem una nova funció que s'encarregarà d'escriure el link en el lloc on vagi el dia.

Aquesta nova funció es diu **escriuDia(**pAny, pMes, pDia**)** i se li han de passar els paràmetres corresponents al dia, mes i any que s'està escrivint a sobre el calendari. Aquesta funció s'encarregarà d'escriure el dia i a més deixarà preparat el 'link' per poder retornar la data elegida a la finestra mare.

Per referenciar la finestra mare des d'una finestra filla s'utilitza la propietat **opener** de l'objecte Window. Aquesta propietat conté el nom de la finestra mare.

La nova funció serà la següent:

```
function escriuDia(dia,mes,any)
{
   ++mes;
   return "<a href=' ' onClick='opener.document.form1.data.value=\"" +
                     dia +"-"+ mes +"-"+ any +"\"; return false;'>" + dia + "</a>";
}
```

En aquesta nova funció, s'ha d'incrementar el valor del mes, ja que internament a la funció *calendari()*, la variable *mes* es troba dins el rang entre 0 i 11. La funció retorna una cadena amb el codi HTML necessari per fer el 'link' desitjat, conjuntament amb el número de dia.

Si es desitgés donar una nova funcionalitat als links dels dies que formen el calendari, solament caldria modificar el contingut de la funció **escriuDia()**.

Per incorporar la nova funció *escriuDia()* dins la funció *calendari()*, modificarem i/o afegirem les línies següents a la funció *calendari()* :

Modificar:

```
31c cal += "><center>";
```

Afegir:

```
31d cal += escriuDia(i,mes,any);
31e cal += "</center></td>";
```

Per provar el funcionament s'ha de crear una pàgina HTML que permeti enviar i rebre paràmetres com la que es veu a continuació. Dins el bloc <head>, a part de les funcions *calendari()* i *escriuDia()*, també hi ha la funció *popup()*.

La funció **popup()** s'encarrega de crear la finestra que contindrà el calendari, a la **pràctica 4 del mòdul 3** s'explica com obrir finestres filles des d'una finestra mare.

Dins el bloc <body> es crea un formulari, de nom *form1*, amb tres camps de text i dos botons. En els dos primers camps de text s'introdueix el mes i any a enviar a la funció. El tercer camp de text rebrà el valor al que hem fet clic en el 'link' del calendari.

Quan fem clic sobre el botó "Crear cal.", s'invoca la funció **popup()** mitjançant el seu controlador d'event **onClick**, passant-li a la vegada el valor del mes i any. La funció *popup()* és la que crearà la finestra filla i a la vegada invocarà la funció *calendari()* passant-li els mateixos paràmetres.

El codi HTML és el següent:

```
<html>
<head>
<title>Calendari</title>
<script language="JavaScript">

  var finestra;

    .... afegir la funció calendari() .....

    .... afegir la funció escriuDia() .....

function popup(pAny, pMes)
```

```
{
  window.focus();
  opt = 'resizable=1,scrollbars=1,width=300,height=225,left=5,top=60';
  finestra=window.open('', 'popup', opt);
  with (finestra.document) {
    writeln('<html>\n<head>\n<title>Calendari</title>\n</head>');
    writeln('<body bgcolor="#c0c0c0">');
    writeln(calendari(pAny, pMes));
    writeln('</body>\n</html>');
    close();
  }
  finestra.focus();
}
</script>
</head>

<body bgcolor="#99ffcc">
<form name="form1">
  <table border="0" width="250" align="center">
  <tr>
    <td>Mes (1..12, 0 actual):</td>
    <td align="center"><input type="text" name="mes" size="5" value="0"></td>
  </tr>
  <tr>
    <td>Any (0 actual):</td>
    <td align="center"><input type="text" name="any" size="5" value="0"></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <input type="button" value="Crear cal." onClick="popup(any.value,mes.value);">
    </td>
  </tr>
  <tr>
    <td>Data elegida:</td>
    <td><input type="text" name="data" size="11" value=""></td>
  </tr>
  </table>
</form>
</body>
</html>
```

El temps al meu costat > **Barra de progrés**

Quan instal·lem un programa, fem una còpia d'arxius o carreguem una pàgina des d'Internet, una barra de progrés ens ajuda a saber com va la feina.

A les pàgines web, aquesta pot ser una manera d'indicar que la pàgina no està inactiva, que s'ha engegat un procés i encara no s'ha acabat. També podríem usar un missatge del tipus "Espereu un moment ...", però una barra informa més.

Hi ha moltes maneres de fer barres de progrés, però aquí només en veurem una. Amb tres exemples, ens aproximarem a un estil de programació que genera codi curt i net. La barra és l'excusa. El que compta és l'algoritme i el llenguatge n'és una eina.

Conceptes JavaScript en aquest capítol

Mètodes **setTimeout()** i **clearTimeout() :** creen i eliminen un temporitzador.
Mètodes **setIntervalt()** i **clearInterval() :** creen i eliminen un interval de temps.
Funció **eval() :** avalua una expressió o bloc de codi.
Operador **-= :** resta el valor de la dreta a la variable de l'esquerra i li assigna el resultat.

1.- Barra gràfica

Per començar, farem una barra gràfica amb els elements imprescindibles i poc codi. Donem per suposat que les imatges que la formen són les úniques o, si més no, les primeres que hi ha a la pàgina. No és la millor manera, perquè poques vegades ho tindrem tan fàcil, però ens servirà per entendre'n el funcionament i poder ampliar-la més endavant.

Es tracta de dibuixar una tira d'imatges iguals i, després, substituir-les progressivament per unes altres. Al cos (<body>) de la pàgina farem un bucle que dibuixi les imatges:

```
 for( i=0; i<40; i++ ) {
   document.write( '<img src="images/punt.gif" width="5" height="20">' )
 }
```

Utilitzem com a base un arxiu .gif de 1x1 píxels, "estirat" fins a 5x20. Posem 40 imatges iguals d'aquesta mida amb el mètode **write**, perquè no quedi separació entre elles. Tanquem javascript i fem un parell de vincles, l'un per engegar la barra i l'altre per reiniciar.

A la capçalera, declarem una variable (tm) pel temporitzador. També declarem el número d'imatge (nImg), que ens farà d'índex, i el posem a zero. A continuació fem la funció pinta(), que anirem activant mentre quedi barra per pintar.

Temporitzadors i recursivitat. Dins de la funció, primer "netegem" el temporitzador amb el mètode **clearTimeout()** i després comparem el número d'imatge amb la longitud de la barra. Si encara és menor, canviem la imatge actual per una de nova, incrementem l'índex i fem un temporitzador que torni a engegar la funció al cap d'un temps: `tm = setTimeout( "pinta()", 50 )`.

El mètode **setTimeout()** avalua una expressió o crida una funció al cap del temps especificat al segon paràmetre, en mil·lisegons. La funció ha d'anar entre cometes, si no volem que s'executi immediatament. Només ho fa una vegada.

```
<script language="javascript">
var tm
var nImg = 0
function pinta() {
  clearTimeout( tm )
  if ( nImg<40 ){
    document.images[id].src = "images/barra.gif"
    nImg++
    tm = setTimeout( "pinta()", 50 )
  }
```

```
}
</script>
</head>

<body ...>
<h2>Barra de progrés</h2>
<script language="javascript">
for( i=0; i<40; i++ ) {
  document.write( '<img src="images/punt.gif" width="5" height="20"> ' )
}
</script>
<h3><a href="javascript:pinta()">Començar</a>    
<a href="javascript:location.reload()">Reiniciar</a></h3>
...
```

2.- Millorar la funcionalitat

L'exemple anterior, tot i que aquí funciona, està confeccionat, perquè es basa en l'Array **images**. Si la pàgina conté imatges abans de la barra, no funcionaria.

Tenim l'excusa per provar noves maneres. Farem la mateixa barra amb lleugeres variacions: 25 quadres en lloc de 40 i els colors invertits. També canviarem el temporitzador: en lloc de setTimeout() utilitzarem **setInterval()**.

L'objectiu és fer una barra de progrés que puguem posar en qualsevol lloc. Si baixem les imatges des d'una funció situada a la capçalera: Per cridar-la des d'un lloc del cos (<body>) cal escriure <script language="javascript">base()</script>. També farem diferent el vincle per activar la barra, amb un enllaç fals i l'event onClick="return inicia()".

Deixem el cos i anem a les funcions. La primera, **base()**, pinta l'espai de la barra. Abans era al <body> (sense "function") i ara és al <head>. La novetat més important és que, ara, les imatges tenen un nom, format per la paraula "quadre" i el número d'índex del bucle. Un cop hem passat per aquí, les imatges de la barra s'anomenen "quadre0", "quadre1", ... Amb això, ja ens és igual si la pàgina conté més imatges abans o després.

La funció **inicia()** que hem associat a l'event onClick, ha de retornar fals, això ja ho sabem. Abans, però, crea un temporitzador: tm = setInterval('pinta()', 50), que crida la funció "pinta()" cada 50 mil·lisegons (i continuaria indefinidament si no la parem).

La funció **pinta()** també està canviada. Primer crea el nom de les imatges amb la paraula "quadre" i l'índex (nImg), de forma que ens hi podem referir amb "document." més el nom:

```
eval( 'document.' + nom + '.src = "images/punt.gif"' )
```

La funció interna **eval()**, la "perla" de JavaScript, permet avaluar expressions formades per constants, variables, literals o tots junts, i les executa. En aquest cas, cada vegada el nom de la imatge és diferent (quadre0, quadre1, ...) i per això no el podem posar com a part d'una instrucció. La funció eval() ho soluciona. A cada pas donarà com a resultat:

```
document.quadre0.src = "images/punt.gif",
document.quadre1.src = "images/punt.gif", ...
```

Finalment, haurem d'incrementar el comptador i, si ha arribat a 25, "netejar" el temporitzador perquè no continuï actuant: if (nImg==25) clearInterval(tm). Com passa sovint en JavaScript, el codi és més fàcil d'entendre que l'explicació:

```
<script language="javascript">
var tm
var nImg = 0
function base() {
  for( i=0; i<25; i++ ) {
    document.writeln( '<img src="images/barra.gif" width="8"' )
    document.write( 'height="16" name="quadre' + i + '">' )
  }
}
function inicia() {
  tm = setInterval( 'pinta()', 50 )
  return false
}
function pinta() {
```

```
  var nom = "quadre" + nImg
  eval( 'document.' + nom + '.src = "images/punt.gif"' )
  nImg++
  if ( nImg==25 ) clearInterval( tm )
}
</script>
</head>

<body ...>
<h2>Barra de progrés</h2>
<script language="javascript">base()</script>
<br><br><a href="" onClick="return inicia()">Començar</a>    
<a href="javascript:location.reload()">Reiniciar</a></h3>
...
```

3.- Vertical i a la carta

L'estructura del segon exemple queda més clara, però el temporitzador del primer és millor. Al següent exemple veurem una barra vertical, que s'omple de baix a dalt, i que funciona sense la limitació del número d'imatges, és a dir, s'adapta als passos que hagi de fer. Per comprovar-ho, establirem els passos i el temps d'espera amb un formulari.

Amb el Dreamweaver, fem una taula de dues cel·les. En una hi posarem la barra. L'altra contindrà un formulari amb dos camps de text, ***passos*** i ***retard***, i dos botons, l'un per engegar la barra i l'altre per reiniciar la pàgina. Utilitzem l'event onClick = "return inicia(form1)".

Un cop dissenyat el cos, passarem a les funcions JavaScript de la capçalera. En aquest cas, comencem declarant quatre variables globals:

- **factor** : un número que indica com ha de créixer la barra.
- **temps** : quants mil·lisegons ha d'esperar fins al proper intent.
- **tm** : "temps mort", variable per referenciar el temporitzador.
- **nImg** : número d'imatge que es substitueix a cada pas. S'inicialitza al 21.

Ja coneixem la funció **base()**, que dibuixa la barra. Ara anirà dins d'una cel·la. Cada imatge serà de 40x8 amb un píxel de separació per sobre i per sota (vspace). Com abans, el nom de les imatges es forma amb "quadre" i l'índex del bucle.

Deixem, de moment, la funció de pintar, i fem la funció **inicia(f1)**, que processa el formulari. L'hem passat com a paràmetre i, a dins de la funció s'assigna a una variable, **f1**. També assignem a la variable "temps" el valor del camp "retard", convertit a enter amb **parseInt()**.

Fem el mateix amb els "passos" i els assignem a una variable local, **nPas**. Si és menor que 1, li posem 1, i guardem a la variable "factor" el quocient de dividir 20 entre el número de passos. Com que el divisor ha estat triat per l'usuari, el resultat pot ser decimal. Només ens queda cridar la funció de pintar i retornar "false" al botó del formulari que l'havia activat.

La funció **pinta()** és la peça clau. Tot i que no és complicada, cal analitzar-la detingudament. La farem de manera que pinti la barra independentment del número de passos. Recordem que tenim 20 imatges. Haurem de preveure que els passos siguin menors o majors que aquest número. Com? Mirem i provem un moment el codi, i ho aclarim després.

```
<script language="javascript">
var factor, temps, tm
var nImg = 21
function base() {
  for( i=1; i<21; i++ ) {
    document.writeln( '<img src="images/punt.gif" width="40"' )
    document.writeln( 'height="6" vspace="1" name="quadre' + i + '"><br>' )
  }
}
function pinta() {
  clearTimeout(tm)
  nImg -= factor
  if( nImg>=1 ) {
    if( factor>1 ) {
```

```
      for( i=20; i>nImg-1; i-- ) {
        eval( 'document.quadre' + i + '.src="images/barra.gif"' )
      }
    }
    eval( 'document.quadre' + Math.floor(nImg) + '.src="images/barra.gif"' )
    tm = setTimeout( 'pinta()', temps )
  }
  else document.img1.src="images/barra.gif"
}
function inicia( f1 ) {
  temps = parseInt( f1.retard.value )
  var nPas = parseInt( f1.passos.value )
  if( nPas<1 ) nPas = 1
  factor = 20/nPas
  pinta()
  return false
}
</script>
</head>

<body ...>
<table width="350" cellpadding="3" align="center">
<form name="form1">
  <tr>
    <td align="center" width="42" nowrap bgcolor="#99AA99">
      <script language="javascript">base()</script>
    </td>
    <td align="center" bgcolor="#BBCCBB"><h3>Barra de progrés</h3>
      Passos: <input type="text" name="passos" size="3" value="20"
        maxlength="3">  
      Retard: <input type="text" name="retard" size="3" value="50"
        maxlength="3"> milisegons<br><br><br>
      <input type="button" name="ok" value="Començar"
        onClick="return inicia(form1)">    
      <input type="button" name="reset" value="Reiniciar"
        onClick="location.reload()"><br>
    </td>
  </tr>
</form>
</table>
...
```

La funció **pinta()** comença amb la neteja del temporitzador, clearTimeout(tm), i, a continuació assigna al número d'imatge el valor que tenia menys el del factor: nImg -= factor. És a dir, hem començat per la imatge de sota i anirem restant fins arribar a la primera. Com que "factor" pot ser un decimal, "nImg" també ho pot ser. Aquest detall és important.

Després, comprovem que el número d'imatges és encara major o igual que 1: if(nImg >=1). Si és així, hem de considerar que, potser, haurem de representar amb 20 imatges un número menor que 20, o, dit d'una altra manera, el "factor" pot ser major que 1 if(factor>1), i s'haurà de pintar més d'una imatge cada vegada. Per això, fem un bucle que les pinti des del peu de la barra fins a l'anterior de la que toca:

```
  eval( 'document.quadre' + i + '.src="images/barra.gif"' )
```

Sigui quin sigui el factor, ara hem de pintar la imatge de la barra que correspon al valor actual. Com que "nImg" pot ser decimal, utilitzem el mètode **Math.floor()**, que, com hem vist en altres pràctiques, elimina els decimals i es queda amb l'enter:

```
  eval( 'document.quadre' + Math.floor(nImg) + '.src="images/barra.gif"' )
```

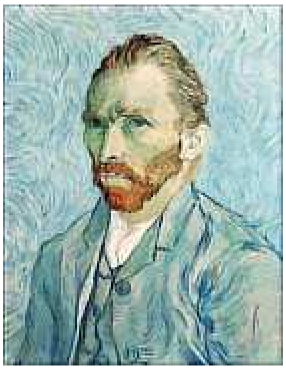
Finalment, només ens queda establir el temporitzador que tornarà a executar la mateixa funció al cap del temps que hagi establert l'usuari: tm = setTimeout('pinta()', temps). Si ja s'ha arribat a la primera imatge, pot ser que el valor de "nImg" sigui menor que 1 i encara no l'hagi pintada. Per això, la pintem directament, sense "eval()", perquè sabem el seu nom:

```
  else document.img1.src="images/barra.gif"
```

Tenim una funció universal, que s'adaptarà a qualsevol necessitat. Hem utilitzat dues tècniques avançades, la recursivitat i l'avaluació de codi. I ens ha quedat curteta. Ho hem trobat difícil? En programació, el que compta és l'algoritme.

El temps al meu costat > **Accions en segon pla**

Què fa la connexió mentre es llegeix la pàgina? Sovint, res. En canvi quan, després, l'usuari fa clic sobre un enllaç, ha d'esperar que l'arxiu arribi i, si és una imatge gran, pot trigar. Amb els temporitzadors de JavaScript i una mica d'estratègia podem aprofitar el "temps mort" en què la connexió està aturada per carregar el que s'ha de veure a continuació.



Això no sempre està ben vist, i s'ha d'utilitzar amb molta cura. Hem d'estar segurs (o gairebé) que l'usuari voldrà veure allò que li colem al seu disc dur sense permís, i que s'estimarà més que fem aquest joc brut que no pas haver d'esperar.

En aquesta pràctica només veurem un exemple: una col·lecció de fotos sobre pintures de Vincent Van Gogh. Quan s'acabi de mostrar la primera imatge, engegarem un procés en segon pla que carregarà les altres, de manera que, quan l'usuari les vulgui veure, les pugui tenir a l'instant.

Tot i que els aspectes gràfics es tracten al mòdul següent, aquí iniciarem el treball amb imatges des de JavaScript. Ens servirà d'aperitiu.

Conceptes JavaScript en aquest capítol

Event **onLoad() :** (aplicat a imatges) executa una acció quan la imatge està carregada.
Objecte **document.images :** Array amb totes les imatges d'una pàgina
Propietat **complete :** determina si una imatge ja està totalment carregada.
Mètode **new Image() :** crea un objecte imatge de JavaScript, buit.
Propietat **src :** adreça de l'arxiu que conté una imatge

1.- Variables globals i estructura

Les accions en segon pla són un concepte genèric que es pot aplicar en molts llocs i de diverses maneres. Aquí plantegem una aplicació d'objectiu didàctic, per veure com es fa, i això condiciona l'estructura. Només és un exemple, l'important és observar l'estratègia.

Utilitzarem JavaScript en dos llocs: a la capçalera <head> hi posarem les variables globals i les funcions, i al cos <body>, el bucle que farà els enllaços amb les fotos. L'aplicació suposa que totes les fotos són a la mateixa carpeta, que tenen format **.jpg**, i que el seu nom és la paraula "vg0", seguida d'un número entre 1 i 9: vg01.jpg, vg02.jpg, ... vg09.jpg.

Comencem amb les variables globals:
- **model** : la primera part del nom de les fotos: images/vg0. S'assigna des del cos.
- **nImg** : el número de la imatge. S'incrementa quan la imatge anterior està carregada.
- **nMax** : número màxim de fotos. S'utilitza per saber quan s'ha d'acabar.
- **tm** : referència al temporitzador.
- **foto** : Array d'objectes imatge. És el que anirem omplint amb les fotos.

També a la capçalera hi escriurem tres funcions, que veurem en detall més endavant:
- **fesImg()** : resol el nom de cada foto, l'afegeix a l'Array i n'informa.
- **tempsMort()** : fa comprovacions i gestiona el temps mort.
- **segonPla()** : rep els paràmetres, assigna les variables globals i engega el procés.

Passem al cos. Posem un formulari amb un camp de text que ens informarà sobre el número d'imatge que s'està carregant. A continuació, situem una imatge, amb la particularitat que assignem l'event onLoad="segonPla('images/vg0','9')". És a dir, quan la foto s'hagi acabat de carregar (onLoad), s'executarà la funció "segonPla" amb dos paràmetres: el model de foto i el número màxim.

Després, obrim JavaScript i fem el bucle dels enllaços. Tot i que no és la forma habitual, aquí hem fet el famós enllaç buit i assignem l'event onClick, de manera que es substitueixi

la foto que es mostra en pantalla: onClick="img1.src=\'' + imatge + '\';return false". Altre cop utilitzem l'antibarra (\) per indicar que el que ve a continuació són unes cometes simples o apòstrof.

```
<script language="javascript">
var model, nImg, nMax, tm
var foto = new Array()

function fesImg() ...

function tempsMort() ...

function segonPla() ...

</script>

<body>
<center>
<form name="form1">
<b>Carregar imatges en segon pla:</b>  
<input type="text" name="actual" size="1" value="0" >
</form>
<img name="img1" src="images/vg01.jpg" border="1" width="360"
height="260" onLoad="segonPla('images/vg0','9')"><br><br> 
<script language="javascript">
for( i=1; i<10; i++ ) {
  var imatge = 'images/vg0' + i + '.jpg'
  document.writeln( '[<a href="" onClick="img1.src=\'' + imatge + '\';return false">' )
  document.writeln( '<b> ' + i + ' </b></a>]  ' )
}
</script>
...
```

2.- Funció segonPla(m, n)

Suposem que s'acaba de carregar la primera imatge. Tal com hem previst, cridarà la funció "segonPla" amb dos paràmetres, el model i el número màxim de fotos. La funció els rep i els assigna a dues variables locals, **m** i **n**. Primer, "neteja" el temporitzador.

A continuació cal comprovar que totes les imatges de la pàgina s'han acabat de carregar. En aquest cas, no caldria, perquè sabem que només n'hi ha una, però és un bon hàbit. Utilitzem una variable ok = true i fem un bucle que repassi les imatges del document. Si alguna encara no s'ha completat, assignem ok = false i sortim del bucle (break).

L'expressió "! document.images[i].complete" utilitza l'operador de negació (!) i consulta la propietat **complete** de la imatge actual.

Si "ok" és encara true és que totes les imatges de la pàgina s'han carregat. Aleshores, assignem les variables globals i fem un temporitzador a la funció "tempsMort()". En cas contrari, tornem a intentar-ho al cap d'un temps utilitzant recursivitat, és a dir, fem un temporitzador a la pròpia funció "segonPla" amb els paràmetres que havia rebut.

```
function segonPla( m, n ) {
  clearTimeout( tm )
  var ok = true
  for( i=0; i<document.images.lenght; i++ ) {
    if( !document.images[i].complete ) {
      ok = false
      break
    }
  }
  if( ok ) {
    model = m
    nMax = parseInt( n )
    nImg = 0
    tm = setTimeout( 'tempsMort()', 100 )
  }
  else tm = setTimeout( 'segonPla("' +m + '","' + n + '")', 100 )
}
```

3.- Funció tempsMort()

Bé, si arribem a aquesta funció és que som en un temps mort, la pàgina ha acabat la seva càrrega d'imatges. Com que hi haurem arribat amb un temporitzador, primer, el netegem, i, seguidament, comprovem si també s'han carregat les fotos en segon pla. Si és així, tornem.

La línia: **if(nImg>=nMax || nImg==nImg-1) return**
Vol dir: *si el número d'imatge ha arribat al màximo és el mateix de la vegada anterior, torna*.

A continuació hem de fer una distinció: si la imatge encara no és la segona, incrementem el comptador i cridem la funció "fesImg()". Si no, abans hem de comprovar que la imatge anterior ja és completa. Si no ho féssim, el navegador intentaria carregar-les totes alhora. En qualsevol cas, la funció, recursiva, acaba cridant-se a ella mateixa amb un temporitzador.

```
function tempsMort() {
  clearTimeout( tm )
  if( nImg>=nMax || nImg==nImg-1 ) return
  if( nImg<2 ) {
    nImg++
    fesImg()
  }
  else {
    if( foto[nImg-1].complete ) {
      nImg++
      fesImg()
    }
  }
  tm = setTimeout( 'tempsMort()', 100 )
}
```

4.- Funció fesImg()

Aquí s'hi arriba en fila. Quan aquesta funció processa la primera foto en segon pla és perquè totes les imatges de la pàgina són completes i, quan processa cadascuna de les altres, és perquè està totalment carregada l'anterior.

És ben simple: crea el nom de la imatge a partir del model, el número i ".jpg". A continuació, fa un nou objecte "Image()" i assigna el nom que ha creat a la propietat **.src**. Amb això el navegador en té prou per començar la càrrega de la foto.

Finalment, com que l'objectiu de l'exemple és didàctic i volem saber què fa la funció, escriu el número d'imatge al camp del formulari que havíem posat per aquest motiu.

```
function fesImg() {
  var nomImg = model + nImg + '.jpg'
  foto[nImg] = new Image()
  foto[nImg].src = nomImg
  document.forms[ 0 ].actual.value = nImg
}
```

5.- Tot plegat

Per veure el codi de l'aplicació, l'obrim amb qualsevol dels botons anteriors i utilitzem el visor de codi font del nostre navegador.

Resumint, tenim una pàgina que carrega una foto i, quan ho ha fet, engega una funció que comprova si queden feines pendents. Quan veu que no en queden, engega una altra funció que carrega una foto en segon pla i, un cop acabada, una altra, fins al màxim que li hem dit.

La principal dificultat de treballar amb temporitzadors és psicològica: les diverses funcions no s'executen en l'ordre en què les hem escrites, sinó quan es "dispara" el temporitzador. Això fa que la lectura del codi resulti difícil de seguir, perquè hem d'imaginar que "s'ha esgotat el termini i aleshores salta a ...". Potser haurem de revisar el codi diverses vegades.

A l'exemple hem posat a tots els temporitzadors un temps d'espera de 100 mil·lisegons. És així perquè escrivim el número en un quadre de text i volem que es vegi. Quan s'aplica a una pàgina "de veritat", normalment pretenem tot el contrari: que no es perdi temps i que no es noti que fem feina en segon pla. Fàcil, traiem el quadre de text i posem 1 mil·lisegon.

Algú pot opinar que no cal tant codi, i que amb un bucle n'hi hauria prou. Això està molt bé per a la precàrrega (ho veurem al mòdul següent) però aquí no. Un bucle deixaria al navegador la feina de gestionar la càrrega, i aquest ho posaria a la seva "cua" de processament. Quan l'usuari fes clic en un vincle, hauria d'esperar el seu torn mentre es carreguen les imatges.

Una última observació: tot això s'engega des de l'event "onLoad" en acabar la càrrega de cada foto. D'aquesta manera, si l'usuari fa clic en un vincle quan encara no s'han carregat totes, es mostra la que demana i es torna a començar el procés de comprovació. Si l'algoritme és correcte, haurem aprofitat el temps mort al màxim. *Time is on my side*.

El temps al meu costat > Temps controlat

A hores d'ara segur que controlem molt millor el que passa a les nostres pàgines, perquè comencem a controlar el temps. Per acabar aquest mòdul, farem un parell d'exercicis que ens obligaran a revisar i consolidar gairebé tot el que hem vist a les pràctiques.

Exercici 1

El primer exercici es basa en les pràctiques 1 i 2. Es tracta de fer un rellotge i un calendari que marquin l'hora i la data actual, indicant el dia de la setmana. Cal que ho fem en un format petit perquè, el seu objectiu hipotètic - o real, si ho volem - seria el de formar part d'una pàgina més gran, com a servei o valor afegit que s'ofereix als usuaris.

Per fer-ho petit i compacte, haurem de jugar una mica amb els estils i les mesures de la taula. Més endavant aprendrem a posar-ho tot dins d'una capa i situar-ho al lloc de la pàgina amb precisió. Ho podem intentar ara amb l'ajut de Dreamweaver o a mà.

Exercici 2

El segon exercici es basa en les pràctiques 3 i 4. Aquí haurem de fer una aplicació - podem prendre com a model l'àlbum de fotos o inventar-ne un altre - que faci una feina en segon pla i vagi mostrant el progrés amb una barra.

Podem utilitzar el tipus de barra que més ens agradi tot i que - cal recordar - els estàndards tenen més acceptació. Tindrem el problema d'haver de gestionar dos temporitzadors. Si ens hi atrevim (no és difícil) podem usar-ne només un, fent que la barra progressi en la mateixa funció que fa la feina en segon pla. Aquesta és la millor opció.

Naturalment, haurem de comprovar que tots dos exercicis funcionin correctament en tots dos navegadors. De vegades (potser ja ho hem experimentat) això és el que costa més.

Finalment, hem d'integrar els exercicis dins l'estructura del nostre curs, al disc dur local i a Internet, de forma que s'hi pugui accedir des de la pàgina d'entrada.

Aquest curs també té projecte

En acabar el mòdul següent (mòdul 6) haurem de plantejar una proposta de projecte i, un cop acceptada, desenvolupar-la.

Per això, abans de començar el mòdul, llegiu les **Recomanacions per al projecte** i feu un cop d'ull als títols dels mòduls següents o, fins i tot, mireu una mica les pràctiques, per poder tenir una visió més acurada dels temes que encara no s'han tractat.

Envieu un missatge a la vostra tutoria un cop l'exercici estigui penjat a Internet. Indiqueu sempre l'adreça de forma que s'hi pugui accedir directament.

L'aspecte visual

Més val una imatge que mil paraules. Tots ho hem sentit dir alguna vegada.

Aquesta és la filosofia màxima de les pàgines web: multimèdia. La informació ens arriba per diferents sentits i fa que sigui més agradable i entenedora per als usuaris.

L'aspecte visual d'una pàgina representa l'esforç del seu creador per facilitar la tramesa d'informació cap a l'usuari. Hem de fer que estigui còmode, i "vegi", més que llegeixi, la informació que li subministrem. Les imatges ens ajuden a fer que el missatge s'entengui més ràpidament.

Però, hem d'anar en compte! No hem d'oblidar mai que el més important de la pàgina és la informació que transmet i no els "efectes especials". Per tant, s'ha de tenir bon gust en l'elecció d'aquests efectes i pensar sempre que són un complement de la informació i mai l'objectiu de la pàgina.

En aquest mòdul analitzarem el tractament de les imatges i els altres elements que introdueixen dinamisme a la pàgina web.

A la **pràctica 1** veurem com podem manipular les imatges des de JavaScript, efectuar substitucions i precàrregues.

A la **pràctica 2** s'introduiran els estils aplicats a capes, veurem com podem canviar propietats de les capes, facilitant la disposició d'objectes dins l'àrea disponible a la pàgina web i, a la vegada, manipular-los des de JavaScript.

A la **pràctica 3** es mostrarà com podem canviar les propietats dels objectes situats sobre la pàgina i fer animacions, moviments iniciats automàticament o a petició de l'usuari que serviran per estructurar les dades sobre el document.

A la **pràctica 4** crearem una de les aplicacions més preuades pels programadors Web: Els menús, amb rollover's, animats i situats en capes.

Veurem, també, que el treball amb els objectes del navegador i les propietats CSS són la principal font d'incompatibilitat amb els diferents navegadors. Si un efecte es considera prou útil per a la majoria d'usuaris, no hem de menysprear-lo, pel fet de no ser totalment compatible.

El millor és detectar el navegador, i si s'escau, oferir la informació mitjançant un efecte alternatiu per a cada navegador, o com a mínim, mostrar un text informatiu de la incompatibilitat. El que hem d'evitar sempre és deixar "penjat" l'usuari amb un error del navegador que desconcerta i fa pensar que la pàgina no està prou ben dissenyada.

En aquest mòdul:

Pràctica 1: **Imatges i scripts**
Pràctica 2: **Estils per a capes**
Pràctica 3: **Animació**
Pràctica 4: **Menús i persianes**

Exercicis: **Efectes animats**

Per fer les pràctiques, obrim el Dreamweaver i ens situem a la "Vista de código", al mig de la secció <body> que, normalment, és a la línia 8. Allí hem d'inserir el codi dels exercicis. Si no tenim activada la numeració de línies, anem al menú *Ver - Opciones de vista de Código - Números de líneas* i l'activem.

El navegador crea un objecte **image** per a cada etiqueta **<img ...>** que troba a la pàgina i els va col·locant a l'Array **document.images**. Podem accedir a cada una a través d'aquest Array, utilitzant la forma

document.images[num].propietat

Per exemple, es poden mostrar les adreces de totes les imatges fent un bucle sobre aquest objecte

JavaScript proporciona, a més, un mètode constructor, **new Image()**, que permet carregar imatges sense haver d'escriure HTML. En tots dos casos, podrem consultar i, depenent del navegador, modificar les propietats de cada imatge. Les instàncies d'aquest objecte poden tenir un identificador i són sensibles a uns events. Són molts recursos, tot plegat.

Conceptes JavaSript en aquest capítol

Constructor **new Image() :** crea una instància de l'objecte imatge, buida.
Propietat **name :** nom de la imatge, si en té.
Propietat **src :** (ja vista) adreça de l'arxiu. És l'única que permet modificar Netscape 4.7.
Propietat **border :** el" borde" de la imatge, si en té.
Propietats **width** i **height :** amplada i altura, respectivament.
Propietats **hspace** i **vspace :** espai buit a l'entorn de la imatge, horitzontal i vertical.
Propietat **complete :** (ja vista) determina si una imatge ja està totalment carregada.

1.- Propietats de les imatges

Un cop s'ha carregat una imatge, des de JavaScript o HTML, s'estableixen les seves propietats. Ja n'hem vistes algunes en pràctiques anteriors, però ara les veurem totes. Bé, totes no. D'una banda ens deixem "lowsrc", que carrega un arxiu alternatiu en baixa resolució i està obsoleta. De l'altra, Internet Explorer mostra (i permet modificar) moltes més propietats. Només cal que obrim una pàgina amb l'**Inspector d'objectes** i mirem document.images[0].

Farem una pàgina amb un parell d'imatges, cadascuna dins d'un vincle. Les imatges han de tenir un nom i el vincle cridarà una funció JavaScript amb aquest nom com a paràmetre:

<a href = "javascript:propImg('nom_de_la_imatge**')">**
<img name="nom_de_la_imatge**" src= . . .**

La funció **propImg()**, que farem a la capçalera de la pàgina, rebrà el nom i crearà una variable local amb l'objecte: var nomImg = document[nom]

A continuació, declarem una variable "txt" que anirem omplint amb text i les propietats. Després de cada propietat escrivim '\n' per tal de forçar un salt de línia dins del text de la variable. No posem la propietat "complete" perquè l'hem vista a l'última pràctica del mòdul anterior i, en aquest cas, sempre és "true". Finalment, enviem una alerta amb el text creat.

```
<script language="javascript">
function propImg( nom ) {
  var nomImg = document[nom]
  var txt = 'Nom: ' + nomImg.name + '\n'
  txt += 'Arxiu: ' + nomImg.src + '\n'
  txt += 'Mides: ' + nomImg.width + ' x ' + nomImg.height + '\n'
  txt += 'Espai horitzontal: ' + nomImg.hspace + '\n'
  txt += 'Espai vertical: ' + nomImg.vspace + '\n'
  txt += 'Borde: ' + nomImg.border + '\n'
  alert( txt )
}
</script>
</head>
```

```
<body ...>
<h2>Propietats de les imatges</h2>
<a href="javascript:propImg('Desert')">
<img name="Desert" src="images/foto1.jpg" hspace="2" border="0"></a>
<a href="javascript:propImg('Marroc')">
<img name="Marroc" src="images/foto2.jpg" vspace="1" border="2"></a>
...
```

2.- Precàrrega

No cal crear una instància en JavaScript de les imatges que ja són a la pàgina, en HTML. De vegades, però, sí que l'haurem de crear si fem precàrrega. I, quan cal fer precàrrega? Doncs, en principi, quan hi hagi rollovers o moltes imatges a la pàgina. Cal tenir en compte que, a més del propi "pes" de cada imatge, el navegador ha d'establir una nova connexió, i això pot fer perdre molt de temps. No es guanya massa si la pàgina té un parell d'imatges grans. En canvi, si n'hi ha moltes de petites, la precàrrega és un gran invent.

En qualsevol cas, utilitzem el mètode constructor:

var foto1 = new Image([amplada] **,** [altura] **)**

Els paràmetres amplada i altura són opcionals. Si no els utilitzem, el constructor els llegeix de la pròpia imatge i els guarda ... quan l'hagi carregada del tot. El constructor, podríem dir, només crea l'estructura, però, de moment, buida. Per omplir-la cal carregar la imatge:

foto1.src = "images/natura01.jpg"

És evident que aquest mètode resulta lent. Si tenim moltes imatges, el millor és usar un bucle que les vagi carregant. Si les imatges tenen el mateix patró i van numerades, en tindrem prou amb un Array. Si no és així, i la majoria de vegades no ho serà, en necessitem dos. El primer contindrà les adreces (si més no, la part "diferent") i el segon, els objectes.

```
// Mètode directe (utilitzant variables)
<script language="javascript">
var foto1 = new Image()
foto1.src = 'images/foto1.jpg'
var foto2 = new Image()
foto2.src = 'images/foto2.jpg'
// ...
</script>

// Array i bucle, per a imatges indexades
<script language="javascript">
var imatge = new Array()
for ( i=1; i<5; i++ ) {
  imatge[i] = new Image()
  imatge[i].src = 'images/foto' + i + '.jpg'
}
</script>

// Utilitzant dos Arrays i un bucle
<script language="javascript">
var arxiu = new Array( 'primera', 'segona', 'tercera', 'quarta' )
var imatge = new Array()
for ( i=0; i<arxiu.lenght; i++ ) {
  imatge[i] = new Image()
  imatge[i].src = 'images/' + arxiu[i] + '.jpg'
}
</script>
```

3.- Intercanvi i rollovers

El que Dreamweaver anomena "imagen de sustitución" i la majoria "rollover" no és més que canviar una imatge per una altra de la mateixa mida, com a resposta a un event, en aquest cas, onMouseOver i el seu company, onMouseOut.

Això es pot fer de diverses maneres i pot tenir altres usos. Per exemple, no és

imprescindible que hagi de canviar la imatge que té el ratolí a sobre. Tampoc ho és que només es pugui canviar una imatge, o que els events hagin de ser, necessàriament, aquests.

> Per què cal precarregar les imatges que hem de substituir dinàmicament? És veritat que si assignem el nom de l'arxiu a la propietat **.src** a la pròpia definició dels events - onMouseOut i onMouseOver -, l'invent funciona, però cada cop que fem una substitució, la nova imatge s'intenta tornar a carregar del servidor. Amb la precàrrega, el navegador sap que té les imatges i les carrega a l'instant.

Del que es tracta és d'escriure una funció com a resposta a un event. Si tenim les imatges de la pàgina identificades, les podrem manipular. Internet Explorer ens permet modificar gairebé totes les propietats, Netscape 4.7 només "src", és a dir, el nom de l'arxiu.

En aquest cas, farem una funció **chgImg()** que no només faci l'intercanvi de la imatge enfocada, sinó que també haurà d'actualitzar una altra imatge. Comencem fent una precàrrega de les 7 imatges: dos botons enrera, dos endavant, les dues fotos que s'han de mostrar quan el ratolí passa sobre els botons i la foto per defecte.

Cridarem la funció des dels vincles, assignada als envents onMouseOut i onMouseOver:

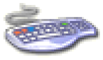
```
<a href="#" onMouseOut="chgImg('ant',0)" onMouseOver="chgImg('ant',1)">
```

Passem com a paràmetres l'identificador de la posició (el mateix que el "name" de la imatge) i el número que correspon a l'índex de l'Array que hem creat a la precàrrega. La funció els identifica com a "id" i "n" i els utilitza per fer l'intercanvi, sempre referit a l'Array:

```
document[id].src = imatge[n].src
```

Aquí s'acaba l'efecte rollover que afecta els botons. Hem intercanviat una imatge per una altra, de la mateixa mida, com a resposta al pas del ratolí. Si no necessitem res més, ja es pot tancar la funció. En aquest cas, però, volem intercanviar una altra imatge en funció de quin sigui el botó sobre el qual ens hem situat, o no. Un condicional i un operador ternari ens ho solucionen.

Hi ha dos estats quan s'analitza el residu de dividir "n" per 2: si és zero (imatges amb índex parell, no som sobre cap botó), assigna la foto per defecte, que té l'índex 6. Si no és zero, assigna la foto 4 o 5, en funció que "n" sigui menor de 2, és a dir, assigna la primera (4) quan som sobre el botó de l'esquerra i la segona (5) quan som sobre el de la dreta.



```
<script language="javascript">
var imatge = new Array()
var arxiu = new Array( 'b_ant1', 'b_ant2', 'b_pos1', 'b_pos2', 'foto1', 'foto2', 'foto3' )
for ( var i=0; i<arxiu.length; i++ ) {
  imatge[i] = new Image()
  imatge[i].src = "images/" + arxiu[i] + ".jpg"
}
function chgImg( id, n ) {
  document[id].src = imatge[n].src
  if (n%2==0) document[ 'foto' ].src = imatge[6].src
  else document[ 'foto' ].src = imatge[ ( (n<2) ? 4 : 5 ) ].src
}
</script>
</head>

<body ...>
<img name='foto' src='images/foto3.jpg'><br><br>
<a href="#" onMouseOut="chgImg('ant',0)" onMouseOver="chgImg('ant',1)">
<img name="ant" src="images/b_ant1.jpg" width="28" height="28" border="0"></a>
<a href="#" onMouseOut="chgImg('pos',2)" onMouseOver="chgImg('pos',3)">
<img name="pos" src="images/b_pos1.jpg" width="28" height="28" border="0"></a>
...
```

En aquest codi hem usat, intencionadament, fotos de diverses mides, i no les hem especificades enlloc. Amb els navegadors que incorporen la versió 2 del DOM, quan es fa l'intercanvi, tot balla. Si, en canvi, ho provem amb Netscape 4.7, la pantalla es manté estable, però les fotos es mostren deformades.

Si no hi ha mesures assignades, l'explorador de Microsoft - que, recordem-ho, permet canviar mesures de forma dinàmica - assigna les originals de la imatge. Navigator 4.7

censidera les propietats **width** i **height** només de lectura, i no permet que els scripts les modifiquin. Cal tenir-ho molt en compte a l'hora de programar.

Si quan posem la foto per defecte <img name='foto' src='images/foto3.jpg'> especifiquem les mides, els navegadors actuals es comportaràn com Netscape 4.7, i les imatges es veuran deformades. En canvi, no hi ha manera d'aconseguir que Netscape 4.7 les mostri amb les seves mides.

Quina és la solució? Doncs, en principi, utilitzant aquest mètode, no n'hi ha. Sempre que posem una imatge en una pàgina, Netscape 4.7 bloqueja les mesures i, si la volem canviar per una altra, o bé haurà de tenir les mateixes mides, o es veurà deformada.

Però, feta la llei, feta la trampa. Si s'han de tornar a escriure les instruccions per crear l'espai d'una imatge, doncs ho fem, dins d'una capa. Però aquesta és una altra història, que veurem més detalladament als mòduls següents.

L'aspecte visual > Estils per a capes

Les capes són un dels elements que permeten donar més dinamisme a les pàgines web. El seu comportament està molt lligat amb les propietats CSS, per tant, és convenient tenir clars aquests conceptes preliminars. La descripció bàsica sobre els estils CSS la podeu trobar a la pràctica 3 del mòdul 4.

Tant l'IE com el NN permeten treballar amb capes, però, malauradament, ho fan de maneres diferents. S'haurà de tenir en compte aquest aspecte per que les pàgines siguin compatibles.

En aquesta pràctica estudiarem quines són les propietats per treballar amb les capes en cadascun dels principals navegadors i implementarem unes funcions que puguin fer compatible el treball amb capes de forma transparent per l'usuari, independentment del navegador emprat.

Elements CSS, HTML i Javascript en aquest capítol.

Directives **<div>** i **<span>** : Defineixen una capa dins el bloc <body>.
Directiva **<style>** : Defineix els estils a aplica a un document HTML.
Objecte **document.all** : Objecte de IE.
Objecte **document.layers** : Objecte de Netscape Navigator versió 4.
Objecte **document.getElementById** : Objecte de Netscape Navigator versió 6.
Propietats **CSS** per a la manipulació de capes.

1.- Propietats CSS per a capes.

Les capes són seccions de la pàgina web, amb contingut HTML, que poden tenir diverses propietats com ara: posició, mida, colors, visibilitat, etc.

Per definir una capa, s'ha d'incloure tot el seu contingut dins les directives **<div>.... <div>** o **<span> ..... <span>**. També existeixen les directives <layer>, <ilayer>, però aquestes solament són reconegudes per NN, per tant no les utilitzarem.

Les capes es poden enniuar, és a dir, es pot crear una capa dins una altra. La seva utilització és la mateixa que a totes les capes, però, part de les seves propietats les pot heretar de la seva capa mare.

Les propietats CSS aplicables a les capes són els següents:

position: Pot ser *absolute* o *relative*. La posició absoluta ofereix les dimensions des de la cantonada superior esquerra de la finestra. La posició relativa pren com a referència l'última posició seqüencial de l'objecte a la finestra del navegador.
left: Distància des del lateral esquerre de la finestra del navegador fins el lateral esquerre de la capa.
top: Distància des de la part superior de la finestra del navegador fins a la part superior de la capa.
width: Indica l'amplada de la capa.
height: Alçada de la capa.
clip: Defineix l'àrea rectangular visible de la capa. S'indiquen els quatre costats de la següent forma: *clip: rect(dalt, dreta, baix, esquerra)*, també pot ser *auto*, on s'agafa la zona per defecte de la capa.
visibility: Indica la visibilitat de la capa. Els valors són: *visible, hidden, inherit*. El valor per defecte és *inherit*, és a dir, hereta la visibilitat de la capa mare. S'ha de tenir en compte que la finestra del navegador també és considerat com una capa.

z-index: Indica l'eix Z, és a dir, la posició Z de la capa. És un valor numèric, les capes amb un
z-index més gran que altres quedaran situades per sobre en el cas de superposició.
background-color: Color de fons de la capa. Amb NN4 aquesta propietat solament aplica el
fons al text. Per a tota la capa, en el NN4 s'ha d'usar la propietat explicada a continuació.
layer-background-color: Color de fons de la capa per a NN4.
background-image: Imatge de fons. Amb el NN4 solament aplica la imatge al text que contingui la capa. Per a tota la capa, en el NN4 s'ha d'aplicar la propietat explicada a continuació.
layer-background-image: Imatge de fons de la capa per a NN4.
overflow: Indica que s'ha de fer si el contingut de la capa no hi cap dins l'àrea definida. Pot ser: *scroll*, *hidden* i *auto*. Amb *scroll* fem aparèixer una barra de desplaçament, amb *hidden* queda inaccessible la part no visible i *auto* permet decidir al navegador quan han d'aparèixer les barres de desplaçament. Aquesta propietat no és reconeguda per NN4.

Per defecte, si no s'indica cap color de fons, la capa quedarà transparent.

A les capes també s'hi poden aplicar les propietats CSS vistes a la pràctica 3 del mòdul 4. En aquesta pràctica no les utilitzarem per facilitar la claredat dels exemples.

2.- Aplicació dels estils per capes.

L'exemple següent ens mostra com hem utilitzat algunes d'aquestes propietats. S'han definit tres capes: "capa1", "capa2" i "capa3". La seva ubicació dins el bloc <body> s'ha fet amb l'ús de les directives <div> per la primera i tercera i <span> per a la segona. A cadascuna d'elles se li ha donat un identificador. El contingut de cada capa és HTML convencional.

Per aplicar les propietats de les capes s'utilitza la directiva **<style>** dins el bloc <head> i referències amb identificador per cada capa. També es podria haver fet, utilitzant el paràmetre "style=..." dins la directiva corresponent, però, ja que el nombre d'estils a aplicar és llarg, queda millor si es defineix dins el bloc <head>.

Podem observar com per a cada capa se li assigna una posició absoluta, és a dir, les referències **left** i **top** preses des de la cantonada superior esquerra de la finestra (origen de coordenades absolutes x=0 i y=0).

Amb les propietats CSS *left* i *top* es posiciona cada capa, les dimensions indicades corresponen a la cantonada superior esquerra de la capa, comptades des de l'origen de coordenades. Amb **width** i **height** indiquem la mida de la capa. Si no s'indica es prendrà com a mida el mínim imprescindible per encabir el seu contingut, tal com passa a la "capa3".

Observeu com s'utilitzen per definir el color de fons les propietats *background-color* i *layer-background-color* amb el mateix color. Això és perquè la primera és interpretada per IE i la segona per NN. La "capa 3" no té definit un color de fons, això vol dir que serà transparent.

El paràmetre **z-index** indica la posició en l'eix Z. La capa que tingui un valor més alt es situarà per sobre de la que té un valor més baix. La "capa1" i la "capa2" no estan sobreposades, per tant no té cap importància el valor de *z-index*. En canvi, la "capa1" i la "capa3" si que estan sobreposades, per tant és necessari que *z-index* de la "capa3" sigui superior al *z-index* de la "capa1" per aconseguir que el text quedi per sobre de la imatge.

A la "capa1" hi ha la propietat **overflow:auto**, en aquest cas, com que el contingut de la capa és més gran que l'espai disponible, apareix una barra de desplaçament vertical. El NN4 no reconeix aquesta propietat.

Proveu de canviar el valor de *visibility* a *hidden*, o els valors de *z-index* de les capes 1 i 3, o el valor de *position* a *relative*, etc i observeu els canvis que es produeixen.

Dins el bloc <head>:

```
...
<style type="text/css">
  #capa1 {position: absolute; left:350px; top:30px; width:200px; height:200px;
```

```
                      visibility:visible; z-index:1;background-color:#B0B1EE;
                      layer-background-color:#B0B1EE;overflow:auto}
  #capa2 {position: absolute; left:50px; top:30px; width:286px; height:402px;
                      visibility:visible; z-index:1;background-color:#FFC0FF;
                      layer-background-color:#FFC0FF}
  #capa3 {position: absolute; left:250px; top:330px; visibility:visible; z-index:2}
</style>
...


Dins el bloc <body>:
...
<div id="capa1">
  <u>Capa 1.</u> Descripció.<br>El <b>Parc Nac...
</div>
...
<div id="capa3">
  <font color="#ff0000" size="38">Aigüestortes</font>
</div>
...
<span ID="capa2">
  <u>Capa 2.</u> Aigüestortes.<br>
  <img src="imatge.jpg">
</span>
...
```

3.- Capes interactives

Les propietats de les capes assignades en temps de disseny també poden ser modificades en temps de visualització, és a dir, mentre es visualitza la pàgina. JavaScript ens servirà per modificar aquestes propietats en temps de visualització.

Ara bé, l'accés a les propietats de les capes no es fa de la mateixa manera segons sigui un navegador IE o NN. A més, el NN tampoc ho fa de la mateixa manera entre el NN4 i el NN6.
Per tant és necessari detectar si el navegador és NN4, NN6 o IE.

En la pràctica anterior ja s'ha explicat com es detecta el navegador. En aquest exemple assignarem tres variables al valor *true* en cas que s'hagi detectat el corresponent navegador. La detecció es fa comprovant si existeixen els objectes "document.layers", "document.all" i "document.getElementById" propis per a cada tipus de navegador.

- ns4 valdrà *true* si és NN versió 4, en cas contrari valdrà *false*.
  `var ns4=document.layers;`
- ie valdrà *true* si és IE, en cas contrari valdrà *false*.
  `var ie=document.all;`
- ns6 valdrà *true* si és NN versió 6 o és IE, en cas contrari valdrà *false*.
  `var ns6=document.getElementById;`

L'accés a les propietats es fa de forma diferent segons el navegador.

- Per a Netscape Navigator versió 4 utilitzem:
  `document.layers.NomCapa.propietat = valor`

- Per a Netscape Navigator versió 6 utilitzem:
  `document.getElementById( ' NomCapa ' ).style.propietat = valor`

- Per a Internet Explorer utilitzem:
  `document.all["NomCapa"].style.propietat = valor` ó
  `document.all.NomCapa.style.propietat =valor` (es pot obviar el text "document.all")

El codi del següent d'exemple s'ha d'afegir al de l'exemple anterior. Fent clic a sobre dels botons, podrem commutar entre capes per mostrar la imatge, el text descriptiu del Parc Nacional d'Aigüestortes, o bé, el seu plànol d'accés.

S'hi han afegit dues capes més:

La capa "**planol**" conté la imatge del plànol d'accés i un botó per commutar les capes de descripció i d'imatge del parc. Si observem les propietats d'aquesta capa en la seva

definició dins la directiva <style> del bloc <head> podem veure que per defecte està oculta, ja que la seva propietat *visibility* val *hidden*.

La capa "**botoactplanol**" solament conté un botó que ha de servir per fer la commutació de capes. La seva ubicació dins de la capa ens és útil per posicionar-lo fàcilment dins la pàgina i ocultar-lo quan s'estigui mostrant el plànol.

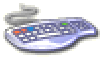
Cada botó activa una funció JavaScript que realitza l'acció esperada: fer la commutació de capes.

El codi JavaScript conté la detecció del tipus de navegador que s'assigna a ns4, ie i ns6. També conté 4 funcions capaON(), capaOFF(), veure() i ocultar().

La funció **capaON()** s'encarrega de posar a *visible* la propietat *visibility* de la capa que se li posa com a paràmetre. Observeu com amb tres "if" es permet accedir a l'acció corresponent segons sigui el navegador. Mitjançant el "eval()" corresponent, es construeix la cadena de codi segons s'ha indicat prèviament i s'executa per modificar la propietat *visibility*. La funció **capaOFF()** fa el mateix, però, posant el valor de *visibility* a *hidden*.

La funció **veure()** s'encarrega de cridar les funcions "capaON()" i "capaOFF()" segons si la capa s'ha de fer visible o no, per mostrar el plànol. El mateix fa la funció **ocultar()** per ocultar la capa del plànol i mostrar les capes: descripció, imatge i nom del parc.

Afegiu el codi següent al codi descrit en l'exemple anterior i podreu comprovar el seu funcionament amb tots tres navegadors.



Afegir al bloc <head> dins de la directiva <style>:

```
#planol {position: absolute; left:120px; top:60px; width:396px; height:299px;
                                                   visibility:hidden; z-index:1}
#botoactplanol {position: absolute; left:500px; top:410px; visibility:visible; z-index:1}
```

Afegir el següent codi JavaScript dins el bloc <head>:

```
<script language="Javascript">
  var ns4=document.layers;
  var ie=document.all;
  var ns6=document.getElementById;

  function capaON(pNomCapa)
  {
    if (ns4) eval("document.layers." + pNomCapa + ".visibility = 'visible' ");
    if (ie) eval("document.all." + pNomCapa + ".style.visibility = 'visible' ");
    if (ns6 && !ie) eval("document.getElementById(' " + pNomCapa + " ').style.
                                                         visibility = 'visible' ");
  }
  function capaOFF(pNomCapa)
  {
    if (ns4) eval("document.layers." + pNomCapa + ".visibility = 'hidden'");
    if (ie) eval("document.all." + pNomCapa + ".style.visibility = 'hidden'");
    if (ns6 && !ie) eval("document.getElementById(' " + pNomCapa + " ').style.
                                                         visibility = 'hidden' ");
  }
  function veure()
  {
    capaOFF("capa1");
    capaOFF("capa2");
    capaOFF("capa3");
    capaOFF("botoactplanol");
    capaON("planol");
  }
  function ocultar()
  {
    capaON("capa1");
    capaON("capa2");
    capaON("capa3");
    capaON("botoactplanol");
    capaOFF("planol");
  }
</script>
```

Afegir dins el bloc <body>:

```
...
<div id="planol">
  <center><img src="planol.jpg">
  <form>
    <input type="button" value="Ocultar plànol" onClick="ocultar();">
  </form></center>
</div>
...
<div id="botoactplanol">
  <form>
    <input type="button" value="Veure plànol" onClick="veure();">
  </form>
</div>
...
```

L'animació dels objectes que es visualitzen en la pàgina web és un dels efectes més apreciats pels dissenyadors web. Existeixen moltes formes de donar dinamisme a la pàgina, utilitzen applets, gifs animats, presentacions Flash, etc. Tots aquests elements indicats són externs al navegador i generalment es carreguen com a plug-in's.

En aquest apartat crearem animació, però, sense utilitzar cap element extern. Únicament utilitzarem els recursos que ens proporciona el mateix navegador, és a dir, manipularem les propietats dels objectes en temps de visualització.

Les propietats dels objectes són fàcilment ajustables en temps de disseny, però, per modificar-los en temps de visualització necessitarem l'ajut de JavaScript.

En aquesta pràctica es presenten dues animacions, una sobre objectes imatge i l'altra sobre capes, però, existeixen moltes més animacions que es podem implementar, l'únic límit està en els recursos que ens subministra el navegador i l'habilitat i imaginació del programador.

Ara bé, el problema més greu serà la compatibilitat entre els diferents navegadors, hi haurà coses que no es podran fer en alguns navegadors i coses que es podran fer en tots però de forma diferent. Haurem de saber en quin navegador es visualitza i si l'efecte no és compatible haurem de mostrar algun altre contingut alternatiu.

Elements del llenguatge en aquest capítol

Mètode **eval()** : Avalua el contingut d'una cadena de text.
Propietat **height** : Conté l'altura d'un objecte.
Propietat **width** : Conté l'ample d'un objecte.
Propietats **left** i **top** : Permeten assignar la posició horitzontal i vertical d'un objecte
Propietat **offsetLeft** i **offsetTop**: Conté la posició horitzontal i vertical d'un objecte (NS6, IE)
Propietat **clientWidht** i **clientHeight :** Conté l'ample i l'alt de la finestra (IE)
Propietat **innerWidth** i **innerHeight** : Conté l'ample i l'alt de la finestra (NS)
Propietat **src** : Assignació del fitxer imatge a un objecte Imatge.
Propietat **visibility :** Indica si l'objecte serà visible o no.
Mètode **Math.round()** : Arrodoneix al sencer més proper un valor numèric.
Mètode **Math.random()** : retorna un valor numèric aleatori entre 0 i 1.
Propietat **zIndex** : Indica la coordenada Z d'un objecte (enter).
Operador **? :** : Operador condicional ternari.
Directiva **<style>** : Permet definir els estils CSS.
Objecte **document.layers** : Objecte de Netscape 4.
Objecte **document.all** : Objecte d'Internet Explorer.
Objecte **document.getElementById** : Objecte de Netscape 6.
Mètode **document.captureEvents()** : Mètode per definir la captura d'events (NN)
Objecte **window.event** : Objecte que conté informació sobre l'últim event (IE)
Events **onmouseup** i **onmousemove**
Propietats **pageX**, **pageY** : retornen la posició del ratolí (a NS)
Propietats **clientX**, **clientY** : retornen la posició del ratolí (a IE)

1.- Commutació d'imatges.

Aquest exemple fa un intercanvi d'imatges fusionant-les entre si. Tot l'efecte consisteix en manipular amb JavaScript les propietats de dos objectes "Image". Les propietats que es manipulen en temps de visualització són l'alçada (**height**), l'amplada (**width**) i la seva coordenada Z (**z-index**).

Les dues imatges estan situades en el mateix espai físic. Mentre una imatge va augmentant la mida, l'altra la va disminuint. Al mateix temps, van alternant la seva

coordenada Z per evitar que una d'elles quedi definitivament amagada a sota.

Aquest efecte no funciona amb NN4. Per tant, si detectem que utilitzem aquest navegador mostrarem un contingut alternatiu. Aquest contingut ha d'ésser una alternativa, per tant, si es considera convenient, es pot substituir per alguna altra imatge estàtica.

Tot i que pot funcionar amb els navegadors IE i NN6, la forma d'implementar-lo és diferent. Ajustar aquesta incompatibilitat és el més complicat del codi.

Dins el codi, a les línies 7-9, detectem el tipus de navegador i dins el bloc <body> creem la pàgina d'una forma o una altra segons sigui el navegador (el codi dins el bloc <body> pot anar conjuntament a un altre codi HTML convencional). Les línies 41-51 creen el contingut, si el navegador és NN4, solament s'escriu un text alternatiu d'avís (línia 42), aquest text es pot substituir per algun altre contingut alternatiu.

Les línies 43 a 50 creen el contingut en cas que sigui un navegador NN6 o IE. Aquest contingut consisteix en definir dues imatges amb identificadors *img1* i *img2* i una capa que conté el botó de commutació. Observeu com les línies 43-50 són la continuació de la cadena iniciada dins de "document.write", per clarificar-ho, s'ha posat la cadena en línies diferents, però, per indicar a l'intèrpret Javascript que la cadena continua a la línia següent, s'afegeix el símbol " \ "en el lloc d'on es parteix.

Les línies 2,3 i 4 defineixen les propietats CSS per a les dues imatges i la capa. La capa **botocambia** no té cap utilitat en l' efecte del programa, solament s'utilitza per facilitar el posicionament del botó de commutació.

L'accés a les propietats dels objectes es fa de forma diferent amb NN6 i IE. Per exemple:

- Per llegir el valor de la propietat *height*, *width* o *zIndex* d'un objecte Image es fa així:
  - Per a IE: document.all.ObjecteImage.propietat
  - Per a NS6: document.ObjecteImage.propietat

- Per assignar un valor a les propietats *height*, *width* o *zIndex* de l'objecte Image es fa així:
  - Per a IE: document.all.ObjecteImage.style.propietat = ...
  - Per a NS6: document.getElementById('ObjecteImage').style.propietat = ...

Com es pot veure són formes totalment diferents de llegir o assignar valors segons sigui el navegador, ja que la manera de referenciar l'objecte és molt diferent. La funció **refObj()** és l'encarregada de facilitar aquestes incompatibilitats. Si la invoquem, ens retorna una cadena per referenciar l'objecte adequadament, segons quin sigui el navegador emprat i segons si volem llegir un paràmetre o el volem modificar (en aquest últim cas hem de passar-li el nom de l'objecte).

La creació de la modificació d'una propietat es fa conjuntament entre la invocació de *refObj()*, que retorna una cadena, i la concatenació de la resta de paràmetres per formar la cadena. Un cop creada tota la cadena, amb l'ajut de la funció **eval()**, s'executa el seu contingut. Tal com es pot veure a les línies 17-21, 25-31 i 34-35

La funció **cambia()** és l'encarregada de fer la manipulació de les propietats, ajudada per la funció *refObj()* per crear les referències dels objectes. Es pren com a referència les dimensions en píxels de les dues imatges, en aquest exemple: img1 és 343x343 i img2 és 227x343 (les dues imatges poden ser de la mateixa mida o diferent). Fixeu-vos com a la línia 3, s'ha indicat per a la imatge 2 un ample i un alt inicial de 0px, és a dir, inicialment la imatge 2 no existeix.

La variable booleana **sentit** representa quina és la imatge que ha de créixer i decréixer en cada moment, inicialment val *true*, això vol dir que la imatge 1 haurà de decréixer i la 2 créixer, un cop feta la transició, s'inverteix el valor d'aquesta variable a la línia 22.

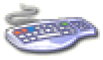
Prenent com a referència la imatge 1, considerem sentit descendent quan aquesta decreix (sentit=*true*) i ascendent quan aquesta augmenta (sentit=*false*) i farem que aquesta modifiqui la seva mida en passos de 5 píxels.

La condició de la línia 17 diu : quan imatge 1 decreixi, si la seva alçada és inferior a 5 píxels o bé quan imatge 2 decreixi i la seva alçada sigui inferior a 5 píxels s'acaba la transició i s'assignen els valors finals a totes dues imatges segons sigui el valor de sentit (línies 18-21).

En cas contrari, (línies 25-28) segons sigui el valor de sentit, s'incrementarà o decreixerà el

valor d'alçada i amplada de la imatge 1 i 2 amb 5 píxels (excepte l'ample d'imatge 2, que es farà en 3 píxels, ja que proporcionalment l'ample d'imatge 2 és inferior al d'imatge 1). També es farà, a cada pas, un intercanvi de la propietat **z-index** per evitar que una imatge quedi definitivament oculta per l'altra (línies 29-36). També es programa un temporitzador (línia 37) per repetir el mateix procés invocant la funció "cambia()", al cap de 10 ms., mentre la imatge que decreix no valgui una alçada inferior a 5 píxels.

El codi a implementar és el següent:



Dins el bloc <head>:

```
1 <style type="text/css">
2   #img1 {position: absolute; left:10px; top:30px; width:343px; height:343px;
               visibility:visible; z-index:1;}
3   #img2 {position: absolute; left:10px; top:30px; width:0px; height:0px;
               visibility:visible; z-index:2;}
4   #botocambia {position: absolute; left:218px; top:380px; visibility:visible; z-index:1}
5 </style>

6 <script language="JavaScript">
7   var ns4=document.layers;
8   var ie=document.all;
9   var ns6=document.getElementById;
10  var sentit=true;
11  function refObj(obj) {
12     if (ie) return (obj!=null)?("document.all." + obj) : ("document.all");
13     if (ns6) return (obj!=null)?("document.getElementById(' " + obj + " ')") :
                 ("document");
14   }
15   function cambia()
16   {
17     if ( (sentit && eval(refObj() + '.img1.height<5')) || (!sentit && eval(refObj()
                 + '.img2.height<5')) ) {
18       eval(refObj('img1') + '.style.width = (sentit)?0:343');
19       eval(refObj('img1') + '.style.height= (sentit)?0:343');
20       eval(refObj('img2') + '.style.width = (sentit)?227:0');
21       eval(refObj('img2') + '.style.height= (sentit)?343:0');
22       sentit=!sentit;
23     }
24     else {
25       eval(refObj('img1') + '.style.width = (sentit)?(' + refObj() + '.img1.width - 5):('
                 + refObj() + '.img1.width + 5)');
26       eval(refObj('img1') + '.style.height= (sentit)?(' + refObj() + '.img1.height - 5): ('
                 + refObj() + '.img1.height + 5)');
27       eval(refObj('img2') + '.style.width = (sentit)?(' + refObj() + '.img2.width + 3):('
                 + refObj() + '.img2.width - 3)');
28       eval(refObj('img2') + '.style.height= (sentit)?(' + refObj() + '.img2.height + 5):('
                 + refObj() + '.img2.height - 5)');
29       if(eval(refObj('img1') + '.style.zIndex==1')) {
30         eval(refObj('img1') + '.style.zIndex=2');
31         eval(refObj('img2') + '.style.zIndex=1');
32       }
33       else {
34         eval(refObj('img1') + '.style.zIndex=1');
35         eval(refObj('img2') + '.style.zIndex=2');
36       }
37       setTimeout("cambia();", 10);
38     }
39   }
40 </script>
```

Dins el bloc <body>:

```
   ...
41 <script language="JavaScript">
42     if (ns4) document.write("Ep! Aquest efecte no esta adaptat per funcionar
                                                  amb NN4");
43     else document.write("\
44          <img id='img1' src='stmaurici.jpg'>\
45          <img id='img2' src='aigtortes.jpg'>\
46          <div id='botocambia'>\
47          <form>\
48            <input type='button' value='Cambia imatge' onClick='cambia();'>\
49          </form>\
```

```
        </div>");
</script>
...
```

2.- Capes animades.

En aquest apartat veurem com podem manipular els paràmetres de les capes en temps de visualització, aconseguint efectes d'animació.

El codi de l'exemple és molt extens (242 línies), per aquest motiu no s'ha inclòs en aquesta pàgina, per veure el codi, heu de visualitzar la finestra de l'exemple i fent clic amb el botó dret del ratolí sobre la finestra, seleccioneu la "Visualització del Codi Font".

Tot i ser un exemple extens, es pot considerar com si fossin quatre exemples diferents d'animació de capes. Els blocs de codi queden clarament diferenciats ja que fan referència a una única capa de les quatre de mostra.

Capa1.- Desplaçament lateral des de l'esquerra a petició de l'usuari.
Capa2.- Capa que simula el comportament d'una finestra de Windows. Inclou una capa enniuada de nom capa2a.
Capa3.- Capa superior que conté el títol, la seva animació s'inicia automàticament en carregar la finestra i es desplaça progressivament des de l'esquerra fins al centre.
Capa4.- Capa que conté una icona (smiley) que es mou constantment per sobre de la pantalla rebotant aleatòriament en els marges. L'animació s'inicia quan la capa3 s'ha posicionat en el centre de la finestra.

Podem aïllar cadascun dels 4 exemples copiant les línies de codi indicades a continuació en una pàgina web a part. En la nova ubicació s'ha de respectar que també estiguin situades dins els corresponents blocs <style> i <script> de <head> i <body>.

- **Comú** a tots els exemples: 14-16.
- **Capa1**: línia 7, línies 17-41 i línies 195-212.
- **Capa2**: línies 8-9, línies 42-112 i línies 213-234.
- **Capa3**: línia 10, línies 113-124 i línies 235-237.
- **Capa4**: línia 11, línies 125-191 i línies 238-240.

El codi indicat dins el bloc <body> i la definició d'estils dins la directiva <style> del bloc <head> apliquen els objectes HTML i els valors de les propietats que es veuen, per defecte, en carregar-se la pàgina. En canvi, el codi, dins la directiva <script> aplica la manipulació de les propietats dels objectes HTML en temps de visualització, de manera que obtenim l'animació desitjada.

No es comentaran les diferents formes d'assignar o llegir valors de les propietats dels objectes ja que tal com està fet el codi es pot interpretar molt fàcilment quina és la forma pròpia de cada navegador. Només cal distingir les línies començades amb if (ie) .... , if (ns4) .... i if(ns6) ... que serveixen per discriminar el codi propi per a cada navegador i interpretar la forma en la qual es llegeix i s'assigna cada propietat.

Part comuna:
La part comuna a tot el codi són les línies 14-16, la seva funció és detectar el tipus de navegador IE, NN4 o NN6 amb què es visualitza la pàgina. Per això es testeja l'existència dels objectes *document.all*, *document.layers* i *document.getElementById* propis de cadascun dels navegadors indicats, respectivament. La variable **ie**, **ns4** o **ns6** valdrà **true** en el cas que sigui el navegador corresponent.

La detecció del navegador és necessària perquè l'accés a les propietats de les capes i tractament d'events són diferents per cada cas. Dins el codi propi de cada capa trobarem condicionals que testegen aquestes variables i apliquen la manipulació de la propietat corresponent amb el format apropiat, en cas contrari obtindríem errors.

Aquesta forma de diferenciar la manipulació de les propietats no és l'habitual en programació JavaScript, en aquest exemple s'ha preferit fer-ho així per facilitar la llegibilitat del codi.

La forma habitual consisteix en la utilització d'una funció, a la qual, passant-li els paràmetres corresponents retorni el codi apropiat segons el tipus de navegador. Posteriorment s'avalua aquest codi amb la funció *eval()*. A l'apartat anterior (exemple de

commutació d'imatges), en tenir un codi menys complex ja s'ha fet d'aquesta forma utilitzant la funció *refObj()*.

Si voleu aïllar el codi corresponent a un sol tipus de navegador, només cal eliminar els blocs de codi que comencen amb if(*tipusNavegador*) ... , on *tipusNavegador* no correspongui amb el que voleu aïllar. La resta de codi és comuna a tots els navegadors.

Capa1:

La capa1 està parcialment oculta en carregar-se la pàgina. Observeu com a la línia 7 s'ha indicat una propietat **left** de -124px. Per aquest motiu queda oculta a l'esquerra. Solament es veu una imatge en forma de botó. La capa conté una taula per facilitar el posicionament del text i la imatge.

La imatge "barra.gif" està dins un link nul (línies 206-208) que detecta l'event onClick. Aquest event invoca a la funció **obreTancaCapa1()**. Observant el codi propi d'aquesta funció (línies 17-41) tenim que inicialment s'ha declarat i assignat una variable global i externa a la funció de nom *estat* amb el valor **false**.

La variable **estat** indica si la capa esta oberta (*true)* o tancada (*false*). Quan s'invoca la funció **obreTancaCapa1()**, el primer que es fa és determinar el valor de la propietat *left* de la capa segons sigui el tipus de navegador (línies 20-22) i s'assigna a **espaiEsq**. Posteriorment es mira el valor d'**estat** (línia 23), si val *false* s'inicia l'apertura de la capa1 (línies 33-39), si val *true* s'inicia el tancament de la capa1 (línies 24-30).

L'obertura i tancament es fa en passos de 5 px, que s'afegeixen o resten (línies 25-27 i 34-36) a la propietat *left* de la capa. A cada increment o disminució efectuat s'activa un temporitzador que tornarà a repetir el mateix procés (línies 28 i 37) fins que ja no es compleixin les condicions (**espaiEsq > -124** i **espaiEsq < 0,** línies 24 i 33). En aquest moment es considera que la capa ja està totalment oberta o tancada, solament queda canviar el valor d'*estat* (línies 30 i 39) i ja s'atura tot el procés fins que l'usuari desitgi tornar a obrir o tancar.

Capa3:

Aquesta és la capa que conté el títol superior. El funcionament de la capa3 és molt similar al de la capa 1, encara que més senzill, perquè només es desplaça en un sentit. A la línia 10 tenim la definició d'estils d'aquesta capa, també podem veure com la propietat *left* està a -400 px, és a dir, totalment amagada a l'esquerra.

Al carregar la pàgina, s'activa automàticament l'event onLoad (línia 194) que invoca la funció **mouCapa3()**. Aquesta funció (línies 113-124) s'encarrega de desplaçar la capa d'esquerra a dreta igual com ho fa la Capa1 quan s'obre.

El desplaçament es fa en unitats de 5px. Cada 100 ms. es torna a invocar la funció (línia 121) i fa un pas més de 5 px, sumant-los a la variable **capa3PosX** que posteriorment s'assigna a la propietat *left* de la capa (línies 117-120).

Quan s'arriba a la posició final (capa3PosX < 10, línia 116) finalitza el posicionament d'aquesta capa i s'inicia l'animació de la Capa4 en invocar-la des de la línia 123. Si no es desitja que s'iniciï l'animació de la Capa 4 solament fa falta eliminar aquesta línia.

Capa4:

Aquesta capa és la que correspon a l' "smiley" que està rebotant aleatòriament per la finestra.
La capa està definida a les línies 238-240. El seu contingut és una imatge de 16x16 px. La imatge forma part d'un link nul. Si feu clic a sobre de la imatge (si ho podeu aconseguir ;-)), s'atura el moviment, ja que s'executa un **clearTimeout()** sobre la variable *sTO* que conté l'identificador del temporitzador d'aquesta capa.

L'estat inicial de la capa es defineix a la línia 11, observeu com té una coordenada Z superior a les altres capes per fer que sempre vagi per sobre d'elles i la seva visibilitat

inicial és oculta.

Quan finalitza el posicionament de la Capa3, s'invoca a la funció **mouCapa4()** (línia 123). La funció *mouCapa4()* (línies 125-191), s'encarrega de moure aleatòriament la capa4 i el seu contingut. El seu moviment són desplaçaments en línia recta fins a trobar un costat de la finestra on rebota. Cada cop que rebota, canvia aleatòriament la seva trajectòria i velocitat.

El primer que fa la funció **mouCapa4()** és determinar la mida de la superfície disponible en el cos del document (línies 129-136) i ho assigna a les variables **ampleBody** i **altBody**. En el cas que el mida de la finestra sigui més gran que el de l'objecte que conté la capa (16x16, línia 137) es determina un valor inicial aleatori **capa4PosX** i **capa4PosY** i es fa la capa visible (línies 140-156).

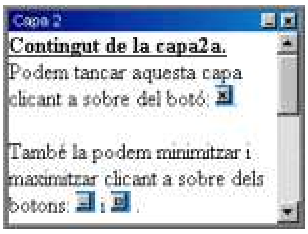
Posteriorment, es determina un valor d'increment X i Y aleatori entre 1 i 11, i un signe d'aquest increment igual a -1 o 1 aleatori (línies 157-160). Finalment es posa la variable **posCapa4** a *false* (línia 161) perquè no es torni a efectuar aquest procés i s'activa un temporitzador per tornar a invocar la funció al cap de 50 ms. (línia 189).

Al tornar-se a executar la funció **mouCapa4()** (per la finalització del temporitzador), es torna a verificar la mida de la finestra i es va directament a la línia 164 on es comprova que la capa estigui dins l'àrea de la finestra (línia 164).

Si s'està dins la finestra, es verifica si l'increment de coordenades portarà la capa fora de la finestra, si és així, es calcula de nou, aleatòriament els valors de l'increment i el signe de desplaçament (línies 165-172). Un cop es té un valor que no ens porti fora de la finestra, s'incrementa a **capa4PosX** i **capa4PosY** i posteriorment s'assigna a les propietats **left** i **top** de la capa4 (línies 173-186).

Globalmentl, l'efecte mostra una capa que es mou de forma rectilínia dins la finestra, amb uns increments i signe de X i Y constants (velocitat i angle constants), però que quan arriba al límit de la finestra, es tornen a calcular aleatòriament els increments i signe de X i Y, canviant la velocitat i l'angle aleatòriament respecte a la trajectòria anterior.

Capa2:



La capa 2 és la més complexa de totes. Per aquest motiu s'ha deixat pel final. De fet està formada per dues capes la **capa2** i la **capa2a**. Aquesta animació no és compatible amb Netscape 4, per tant, si detectem aquest navegador s'haurà de mostrar un contingut alternatiu.

En aquesta capa és necessari capturar els events **onmousemove** i **onmouseup**, encara no s'ha explicat res sobre la captura d'events, per tant, no s'explicarà amb detall. En la pràctica 1 del mòdul 7 s'explica com es capturen els events. Quan hagueu fet aquesta pràctica, podeu tornar a aquest apartat i analitzar amb més detall com es fa aquesta capa.

La definició de les propietats d'aquestes dues capes es fa a les línies 8 i 9, observeu com la *capa2a* té un posicionament relatiu. Com que està situada dins la *capa 2* les seves referències **top** i **left** tindran com origen de coordenades les de la *capa 2*. La visibilitat de la *capa2a* s'hereta de la capa2 i si el seu contingut és extens apareixeran les barres de desplaçament (**overflow:auto**).

Aquestes dues capes es posicionen dins el bloc <body> a les línies 213-234. Observeu com la seva inclusió es fa a partir de codi JavaScript. És necessari detectar el navegador, ja que si es tracta de NN4 no podem crear la *capa2a* perquè es veuria malament (línia 219). Per als altres dos navegadors IE i NN6, es crea el contingut de la capa a les línies 221-231.

La variable de text **txt**, conté tot el contingut de la *capa2* i la *capa2a*. Inicialment s'introdueixen tres imatges corresponents a la capçalera de la capa, són la barra de títol, el botó de minimitzar i el botó de tancar (línies 221-223). Posteriorment es crea la *capa2a* i el seu contingut (línies 224-230).

Les tres imatges de la capçalera responen a events introduïts amb el ratolí. Quan es fa clic a sobre de la imatge (capa2_3.jpg) s'invoca l'event *onClick* i es canvia la propietat de visibilitat de la *capa2* a oculta (línia 223). Si es fa clic a la imatge (capa2_2.jpg),

s'invoca la funció **minMaxCapa2()** (línia 222), observeu que aquesta imatge té un identificador : id="imgMinMax", aquest identificador servirà per poder canviar la imatge a (capa2_4.jpg) a les línies 49 i 54, quan la *capa2* sigui minimitzada. Les línies 62 i 67 fan l'efecte contrari.

La imatge

(capa2_1.jpg) introduïda a la línia 221, també respon a un event, en aquest cas l'event **onMouseDown**. Aquest event invoca la funció **mouCapa2()**, que s'encarrega de capturar els events *onmousemove* i *onmouseup* i permeten desplaçar la capa per sobre de la finestra.

La funció **minMaxCapa2()** (línies 42-71) minimitza o maximitza la *capa2*. Per això utilitza la variable global **minMax** per saber si està oberta o tancada. Per tancar la capa, posa a 0px l'alçada de la *capa2a* i a 23px/19px l'alçada de la *capa2*, segons sigui per a IE o NN6. També canvia la imatge amb identificador *imgMinMax* per mostrar l'altre símbol alternatiu (línies 46-54).

En cas contrari, si s'ha d'obrir la capa, fa el mateix procés, però a la inversa (línies 59-67). Per acabar cadascun dels dos processos (obertura o tancament de la capa) es canvia el valor de la variable **minMax** (línies 56-69).

La funció **mouCapa2()** és la responsable de facilitar el desplaçament. Com ja s'ha dit manipula la captura dels events *onmousemove* i *onmouseup*. A la pràctica 1 del mòdul 7 trobareu més informació sobre aquests conceptes.

La incompatibilitat amb el Netscape 4 bé determinada per la utilització de la *capa2a* enniuada dins la *capa2*. Aquest fet, més la propietat *overflow* assignada a la *capa2*, fa que sigui incompatible amb el NN4. Si aquest mateix exemple no contingues la *capa2a* podria funcionar perfectament amb el NN4 ja que el sistema de captura d'events és el mateix que s'ha aplicat pel cas del NN6.

Per veure el codi d'aquest exemple, s'ha de visualitzar la finestra de mostra i fer clic a sobre de la finestra amb el botó dret del ratolí. Al menú contextual que apareix, heu de seleccionar l'opció de "Visualització del codi font".

Els menús, de tots tipus, mides i colors, són una part de la interfície dels programes que ens resulta molt familiar. Això fa que siguin un recurs oportú per a les pàgines web, on l'usuari entra per primer cop sense saber què es trobarà. Si li oferim solucions que reconegui a primera vista, la visita li resultarà més efectiva i gratificant.

S'anomenen "persianes" les capes en moviment que apareixen, generalment, en sentit vertical. Poden contenir qualsevol element, també un menú. La seva utilitat es fonamenta en un millor aprofitament de l'espai. També en la possibilitat de mantenir una informació amagada i mostrar-la només quan es necessita. Finalment, tenen una funció estètica i donen imatge de dinamisme.

Conceptes JavaScript en aquest capítol

Mètode **split() :** crea un Array a partir d'una cadena, partint-la pel caràcter especificat.
Propietat **innerHTML :** assigna el contingut d'una capa de forma dinàmica. Només iExplorer

1.- Menú amb rollover

Volem fer un menú on cada línia comenci per una imatge que es substitueixi per una altra quan passa el ratolí. Això ho vam veure a la pràctica "Imatges i scripts". Ara, però, ho volem automatitzar amb una mica de JavaScript.

De fet, és el mateix menú que vam fer a l'exercici del mòdul 2, i el podem prendre com a base. Com aleshores, comencem fent un Array que tractarem com a parells adreça - títol. Fins aquí és el mateix. Les diferències comencen al bucle.

A cada volta, definim els events **onMouseOver** i **onMouseOut** de manera que canviïn la propietat **.src** d'una imatge. Quina? Doncs, muntem un nom, per exemple, a base d'una lletra i l'índex actual del bucle. És clar que també podem fer precàrrega i utilitzar una funció d'intercanvi, com hem vist a la pràctica 1. Això ho deixem a l'elecció de cadascú.

Ja que hi som, també redefinim l'event **onClick** perquè ens tregui el marquet de punts del vincle. Després posem la imatge per defecte, amb el seu nom (tal com hem fet abans), l'adreça i les mesures. Finalment, escrivim el vincle i tanquem la fila.

```
<script language="javascript">
var menu = new Array(
"m1/index.htm", "Programar la web",
"m2/index.htm", "Estructures JavaScript",
"m3/index.htm", "L'usuari en acció",
"m4/index.htm", "Atenció a la diversitat",
"m5/index.htm", "El temps al meu costat",
"m6/index.htm", "L'aspecte visual",
"m7/index.htm", "Interacció i comunicació",
"m8/index.htm", "Calaix desastre");
var i, url, nom
for ( i=0; i<menu.length; i+=2 ) {
  url = menu[i]
  nom = menu[i+1]
  with( document ) {
    writeln( '<tr><td class="menu"><a href="' + url + '"')
    writeln( 'onMouseOver="m' + i + '.src=\'images/bola2.gif\'"' )
    writeln( 'onMouseOut="m' + i + '.src=\'images/bola1.gif\'"' )
    writeln( 'onClick="window.focus()">')
    writeln( '<img name="m' + i + '" src="images/bola1.gif" border="0"' )
    writeln( 'width="12" height="12" hspace="4">' )
    writeln( nom + '</a></td></tr>' )
  }
}
</script>
```

2.- Esquema desplegable

Com a segon plat, prepararem la base del que podria arribar a ser un menú més complet: un esquema desplegable. Per tal d'entendre millor l'algoritme, el deixarem pelat. Si algú s'anima i ho vol combinar amb intercanvi d'imatges i amanir-ho amb una mica de bon gust a l'hora de dissenyar els estils, pot quedar força bé.

Comencem aplicant la lògica a la construcció d'un Array. Cada element serà una carpeta i, com que té una certa complexitat, els declararem d'un en un. A la carpeta, que tindrà un títol, hi haurà els submenús, formats per parells adreça - nom. Utilitzarem el símbol punt i coma (;) per separar submenús i la barra vertical per separar els elements del parell:

```
carpeta[0] = 'Primer menú ; sub1.htm | Primer vincle ; sub2.htm | Segon vincle ; ...'
```

És clar que, si ho escrivim en línies separades, s'entendrà millor. De moment, cada element té el títol i els parells. Si volem més submenús només ens cal afegir (o inserir) més parelles. També necessitem un Array, paral·lel a l'anterior, que recordi l'estat de cada carpeta. El declarem i omplim de valors false. D'entrada, totes les carpetes queden tancades:

```
var obert = new Array()
for ( i=0; i<carpeta.length; i++ ) {
  obert[i] = false
}
```

Ja tenim les dades. Ara necessitem una funció per processar-les. **obreTanca(n)** rep com a paràmetre el número de carpeta. Més endavant farem que els títols de les carpetes siguin vincles cap a aquesta funció. Això vol que, quan s'inicia la funció, és perquè s'ha fet clic sobre una carpeta i, en conseqüència s'ha de canviar d'estat: obert[n] = ! obert[n]

A continuació declarem la variable **txt** buida. L'anirem omplint de codi HTML que, en acabar, bolcarem en una capa. De moment, ja sabem què toca: engegar un bucle que faci el recorregut per les carpetes. A cada volta, es troba amb una autèntica "botifarra" de dades, que haurem de partir pel lloc correcte. Com? Localitzant els caràcters separadors, els punt i coma: var item = carpeta[i].split(';')

El mètode **split()** crea un Array a partir d'una cadena, utilitzant com a "punts de tall" el caràcter que li especifiquem. En el cas que ens ocupa, ara tenim un array "ítem" de tres elements: el primer, ítem[0], és el nom de la carpeta i, els altres, les parelles de submenús. Haurem d'afegir al text (variable txt) de la capa el vincle per obrir i tancar la carpeta:

```
txt += '<b><a href="javascript:obreTanca(' + i + ')">' + item[0] + '</a></b><br>'
```

Si la carpeta és oberta, fem un altre bucle que vagi repassant les parelles, les parteixi per on trobi la barra vertical i faci els enllaços. Comencem per l'índex 1 (el 0 era la carpeta):

```
for ( j=1; j<item.length-1; j++ ) {
   var url = item[j].split( '|' )
   txt += '   <a href = "' + url[0] + '">' + url[1] + '</a><br>'
}
```

Un cop acabats els bucles hem d'escriure el text generat a la capa. Ja sabem que els dos exploradors ho fan diferent (mireu el codi). Finalment, hem d'activar el sistema al <body> quan es carrega la pàgina (onLoad), amb el número del menú que vulguem obrir per defecte, i crear una capa per posar el menú. Tindrà, evidentment, l'identificador que hem utilitzat per escriure el text que ha generat la funció, en aquest cas, id="menu".

```
<script language="javascript">
var carpeta=new Array()
carpeta[0]='Primer menú;'+
    'sub11.htm|Primer submenú;'+
    'sub12.htm|Segon submenú;'
carpeta[1]='Segon menú;'+
    'sub21.htm|Submenú 2.1;'+
    'sub22.htm|Submenú 2.2;'+
    'sub23.htm|Submenú 2.3;'
var obert = new Array()
for ( i=0; i<carpeta.length; i++ ) {
```

```
  obert[i] = false
}
function obreTanca(n) {
  obert[n] = !obert[n]
  var txt=''
  for ( i=0; i<carpeta.length; i++ ) {
    var item = carpeta[i].split(';')
    txt += '<b><a href="javascript:obreTanca(' + i + ')">' + item[0] + '</a></b><br>'
    if ( obert[i] ){
      for ( j=1; j<item.length-1; j++ ) {
        var url = item[j].split('|')
        txt += '   <a href="' + url[0] + '">' + url[1] + '</a><br>'
      }
    }
  }
  if (document.all) menu.innerHTML=txt
  else {
    document.menu.document.write(txt)
    document.menu.document.close()
  }
}
</script>
</head>

<body onload="obreTanca(999)">
<div id="menu" style="position:absolute;top:20px;left:30px"></div>
...
```

A l'exemple podem veure una combinació d'aquest exercici i l'anterior, és a dir, un esquema desplegable amb rollover. Mireu el codi font per a més detalls.

3.- **Persiana** (Internet Explorer)

Farem un menú amb l'estil del d'inici dels Windows, en una capa que s'obre cap amunt. Haurem de combinar scripts, css i HTML i només funcionarà amb Internet Explorer. Les diferències entre tots dos navegadors, en el tractament d'estils i capes, són enormes. Si volem compatibilitat, és més pràctic i curt escriure dos arxius diferents. A més, el codi de cadascun ja és prou complex, de manera que, així, l'entendrem millor.

Al cos de la pàgina farem una capa amb identificador <div id="Layer1">, hi escriurem els vincles i la tancarem. A sota, en una altra capa, hi posarem un formulari amb un parell de botons que pujaran la persiana a diferents velocitats.

```
<input type="button" value=" Lent " onClick="Persiana( true, 'Layer1' , 2 )">
```

En el full d'estils establim la lletra del cos (body) - Explorer ho permet -, el color i decoració dels vincles (a), el comportament rollover (a:hover) i les característiques de la capa (#Layer1): posició absoluta, 115x115, z-índex 1, visibilitat oculta, bordes amb relleu, fons gris, lletra de 14, interlineat de 20 i marge intern de 5, tot píxels.

Fixem-nos en el coixinet (#) que utilitzem per definir els estils de la capa. Quan s'utilitza aquest signe, css interpreta que ens referim a una etiqueta concreta (un objecte) i no una classe.

I ara, JavaScript. Comencem declarant una variable global buida: (var obj='') i, a continuació, la funció persiana, que, com hem vist, rebrà tres paràmetres: estat, identificador de capa i velocitat. A dins, haurem d'actuar segons l'estat (variable "inici").

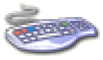
Quan hem d'inicialitzar la capa, l'assignem a la variable global "obj" i li afegim propietats:
  - **obj.y** contindrà l'altura de la capa i ens servirà per fer càlculs.
  - **obj.clp** serà la zona que es mostra, en forma de tant per cent. L'iniciem a zero.
  - **obj.sub**, establert en 140, és la coordenada vertical d'inici de la capa, el seu "borde" superior.
  - **obj.incr** serà l'increment, que arrodonim multiplicant altura per velocitat i dividint per cent.

Si la capa ja estava inicialitzada, el que toca és incrementar la zona que es mostra - utilitzant "obj.incr" - i vigilar que no passi de 100.

Un cop acabada aquesta distinció, redefinim la zona visible (obj.style.clip) a partir del valor de "obj.clp", en forma de %. També calculem la posició superior d'inici de la capa:

obj.style.posTop = Math.round(obj.sup - obj.y * obj.clp / 100)

Si estàvem inicialitzant la capa, ara és el moment de fer-la visible. Finalment, comprovem si "obj.clp" és encara menor que 100. Si ho és, fem un temporitzador que torni a cridar la funció, amb el paràmetre inici a "false" i els altres dos, buits. Si, en canvi, "obj.clp" ja no és menor que 100, el posem a zero i acabem la funció



```
<script language="JavaScript">
var obj=''
function Persiana( inici, capa, vel ) {
  if ( inici ) {
    obj = document.all[capa]
    obj.y = obj.offsetHeight
    obj.clp = 0
    obj.sup = 140
    obj.incr = Math.round( obj.y * vel / 100 )
  }
  else {
    obj.clp += obj.incr
    if ( obj.clp > 100 ) obj.clp = 100
  }
  obj.style.clip = 'rect( auto, auto, '+ obj.clp+'%, auto )'
  obj.style.posTop = Math.round( obj.sup - obj.y * obj.clp / 100 )
  if ( inici ) obj.style.visibility = 'visible'
  if ( obj.clp < 100 ) setTimeout( 'Persiana( false, "", "")', 5 )
  else obj.clp = 0
}
</script>
<style type="text/css">
body { font-family: Arial, Helvetica, sans-serif}
a { color: #000000; text-decoration: none}
a:hover { color: #990000; font-weight: bold; text-decoration: underline}
#Layer1 { position: absolute; width: 115px; height: 115px; z-index: 1;visibility: hidden;
border: 1 solid black; border-left: 1 solid #FFFFFF; border-top: 1 solid #FFFFFF;
background:#D0D0C8; font-size: 14px; line-height: 20px; padding: 5px}
</style>
</head>

<body ... >
<div id="Layer1">
<a href="enlloc1.htm" target="_blank"><b>P</b>rimer vincle</a><br>
<a href="enlloc2.htm" target="_blank"><b>S</b>egon vincle</a><br>
<a href="enlloc3.htm" target="_blank"><b>T</b>ercer vincle</a><br>
<a href="enlloc4.htm" target="_blank"><b>Q</b>uart vincle</a><br>
<a href="enlloc5.htm" target="_blank"><b>C</b>inquè vincle</a>
</div>
<div style="position:absolute; top: 144px">
<form name="f1">
<input type="button" value=" Lent " onClick="Persiana( true, 'Layer1', 2 )">
<input type="button" value=" Ràpid " onClick="Persiana( true, 'Layer1', 8 )">
</form>
</div>
...
```

En aquest mòdul hem vist diverses aplicacions per treballar amb un dels aspectes més importants de les pàgines web, després de la informació: l'entorn visual.

En aquesta pràctica treballarem aquestes aplicacions.

Els exercicis a realitzar són dos, a elegir entre els tres que es proposen a continuació.

Abans de començar modificarem la pàgina *index.htm* que tenim dins la carpeta *m6* i li introduirem dos enllaços, cadascun d'ells ens portarà a un dels dos exercicis que hem elegit.

Exercici 1. Rellotge reubicable.

Aquest exercici és una modificació de l'exercici 1 del mòdul 5.
Es tracta de crear una capa animada (podeu extreure'n una de les que formen la pràctica 3 apartat 2) o una capa estàtica com les de la pràctica 2 que mostri la data i hora actual en el seu interior.

Exercici 2. Imatge amb rollover

Es tracta de crear una pàgina en la que hi hagi una imatge visible. A la vegada, s'ha de fer una precàrrega d'una segona imatge. Posteriorment, quan es passi el ratolí sobre la imatge visible s'ha d'intercanviar per la imatge que hem precarregat.

Fins aquí hem descrit el que és habitual en un rollover. Però en aquest exercici, la imatge original no ha de retornar quan el ratolí surt, sinó quan el ratolí torna a entrar a l'àrea ocupada per la imatge.

Exercici 3. Menú.

Es tracta que implementeu un dels menús vistos en la pràctica 4.

Envieu un missatge a la vostra tutoria un cop l'exercici estigui penjat a Internet. Indiqueu sempre l'adreça de forma que s'hi pugui accedir directament.

Interacció i comunicació

Tot i que ja hem vist diversos aspectes de la interacció i la comunicació, en aquest mòdul hi entrem de ple. De fet, això és el que ens vam proposar a l'inici del curs. Si fem DHTML i JavaScript és, sobretot, per aconseguir pàgines més interactives i millorar la comunicació - o l'intercanvi d'informació - amb els usuaris.

Per poder interactuar hem de saber què passa. El navegador ho sap, i JavaScrpt pot tenir accés a aquesta informació. Cada cop que es mou el ratolí, es fa clic en un vincle, es prem una tecla, es carrega una pàgina o s'abandona, ... es produeix un event, que podem analitzar i preparar una acció com a resposta.

En aquest mòdul veurem els events i el seu tractament: els comportaments. Amb això farem que la web estigui "més atenta" al que passa. Tractarem amb especial atenció la validació de formularis, que ens permetrà, d'una banda, assegurar la consistència de les dades que s'envien i, de l'altra, orientar els usuaris de manera que els puguin utilitzar millor.

La pàgina pot respondre de diverses maneres. Podem, per exemple, aconseguir que els formularis ens enviïn més informació de la que mostren, utilitzant camps ocults. També ens pot interessar restringir l'accés a algunes zones (pàgines) de la web, de manera que només hi pugui accedir qui conegui la clau. O desactivar el botó dret del ratolí, de manera que no ens puguin copiar el codi o robar les imatges ...

Un altra qüestió interessant és la interacció entre marcs. Mentre programàvem només en HTML, els marcs presentaven més problemes que avantatges. Amb JavaScript, però, els podem mantenir sincronitzats. I, encara més; veurem que tots els elements d'un marc són accessibles i manipulables des dels altres, no només els de la pàgina - imatges, vincles, també les variables i les funcions, és a dir, tot. Els marcs, que potser ja donàvem per morts, ara tornen a aixecar el cap, més vius que mai.

En definitiva, aquest mòdul tracta de les pàgines interactives, on l'usuari essent com és el centre d'atenció i qui maneja la pàgina, el traiem del pati de butaques i el fem pujar a l'escenari. I el nostre paper? Doncs, nosaltres, dirigim l'obra a distància.

A partir d'aquí comença, també, el nostre treball sobre el projecte. Les pràctiques següents han de servir per donar-li un caire més d'eina i no tant d'aparador. Hem de procurar que les pàgines "facin coses" i, per això, han d'estar atentes al que passa al seu entorn i tenir una resposta preparada.

En aquest mòdul

Pràctica 1: **Events i comportaments**
Pràctica 2: **Validació de formularis**
Pràctica 3: **La pàgina respon**
Pràctica 4: **Interacció entre marcs**

Exercicis: **Projecte interactiu**

Quan escrivim codi JavaScript, cal recordar que Dreamwever disposa d'una eina, el depurador, que ens facilita la detecció i el tractament d'errades. Es pot activar des del menú "Archivo - Depurar en navegador" o fent clic al botó de la barra d'estat "Vista previa/depurar en explorador".

Interacció i comunicació > Events i comportaments

Quan s'utilitza DHTML tots els elements d'una pàgina poden tenir events associats, que el navegador utilitzarà per executar funcions, és a dir, tots els elements poden fer accions.

Els events es desencadenen de diverses maneres. Ja n'hem vist algunes:
- Accions del navegador: entrar o sortir d'una pàgina, carregar imatges, detectar errades,...
- Accions d'usuari: fer clic o moure el ratolí, prémer una tecla, ...
- Accions del rellotge: els dos tipus de temporitzadors

Els elements d'una pàgina "disparen" els events sempre. El que nosaltres podem fer és dirigir les accions que es fan per defecte, cap a funcions que puguem controlar. Així, quan en HTML escrivim onclick="..." o bé onmouseover="...", el que fem és dirigir la sortida de l'event cap a on ens interessa. Les funcions - que, evidentment, haurem d'escriure - s'anomenen "manejadors d'events" o "Event Handlers". Dreamweaver ho tradueix per "Comportaments", i així és com les anomenarem.

Un exemple de com funciona tot plegat l'hem vist diverses vegades. Els vincles tenen, per defecte, l'event Click associat a una funció del navegador que llegeix la propietat href i l'intenta carregar. De vegades (per exemple, als enllaços buits que tant hem usat), però, no volem que s'executi aquesta funció, i dirigim l'event Click a un altre lloc que retorni "false". O sí que volem que s'executi, però, abans, volem fer una altra cosa.

Ja vam parlar sobre els events al mòdul 3, i els hem utilitzat moltes vegades. En aquesta pràctica els presentarem de forma més detallada i veurem noves possibilitats.

Conceptes JavaScript en aquest capítol

Propietat **this :** referència a l'objecte que provoca un event o acció.
Objecte **event :** interfície JavaScript per manejar events.
Mètode **captureEvents() :** habilita un objecte per respondre a un tipus d'event (Netscape).

1.- Llista d'events

Els navegadors controlen i manegen tot el que passa a través dels events, però cadascun ho fa a la seva manera, com anirem veient. La llista següent no és, doncs, massa segura i, en tot cas, intenta mostrar els mínims:

Event	S'aplica a ...	Passa quan ...
Abort onabort	imatges	L'usuari interromp la càrrega d'una imatge
Blur onblur	finestres i elements de formulari	El component deixa d'estar enfocat o perd el cursor
Change onchange	camps de text i llistes de selecció	Ha canviat el valor del camp
Click onclick	botons, controls de radio i verificació, enllaços	L'usuari ha fet clic sobre l'element
DragDrop ondragdrop	finestres	L'usuari fa clic i arrossega un objecte
Error onerror	imatges i finestres	La càrrega d'una imatge o pàgina produeix una errada
Focus onfocus	finestres i elements de formulari	L'element queda enfocat o rep el cursor
KeyDown onkeydown	pàgines, imatges, vincles i àrees de text	L'usuari prem una tecla
KeyPress onkeypress	pàgines, imatges, vincles i àrees de text	L'usuari prem i deixa anar una tecla
KeyUp onkeyup	pàgines, imatges, vincles i àrees de text	L'usuari deixa de prémer una tecla

Load onload	imatges i pàgines	Es carrega una pàgina o imatge
MouseDown onmousedown	pàgines, botons i vincles	L'usuari prem un botó del ratolí
MouseMove onmousemove	per defecte, no s'aplica enlloc	L'usuari mou el ratolí
MouseOut onmouseout	mapes d'imatge i vincles	El cursor del ratolí surt del mapa o vincle
MouseOver onmouseover	vincles	El cursor es situa sobre un vincle
MouseUp onmouseup	pàgines, botons i vincles	L'usuari deixa de prémer un botó del ratolí
Move onmove	finestres	La finestra es mou
Reset onreset	formularis	L'usuari esborra un formulari
Resize onresize	finestres	La finestra canvia de mida
Select onselect	camps de text	L'usuari selecciona en un camp d'entrada de text
Submit onsubmit	formularis	S'envia un formulari
Unload onunload	cos de la pàgina	L'usuari abandona la pàgina

2.- Definir comportaments

Un comportament, segons Dreamweaver, no és més que una funció que s'executa com a resposta a un event. N'hem vist en diverses pràctiques.

Podem definir el comportament davant d'un event de dues maneres:

- Des de HTML: <img src="bola.gif" onmouseover = "ctrlEv()">
- Des de JavaScript: document.images[0].onmouseover = ctrlEv

La definició d'un comportament es fa com qualsevol altra funció. Els comportaments poden rebre paràmetres. En tots els casos, però, hi ha un paràmetre especial, **this**, que s'assigna sempre i que serveix per referenciar l'objecte que ha provocat l'event. Si volem evitar que també s'executi el comportament per defecte, hem de retornar "false".

A l'exemple següent fem una funció que s'activa des de diversos llocs: quan es carrega la pàgina o quan es fa clic en un vincle. També hem posat una imatge amb un event **onmouseover**. Si ho provem amb els navegadors actuals, funciona; amb Netscape 4.7, no.

```
<script language="javascript">
function avisa( txt ) {
  alert( 'Aquesta funció s\'activa des ' + txt )
}
</script>
</head>

<body onLoad="avisa( 'de la càrrega de la pàgina' )">
<img src="images/docs.gif" onMouseOver="avisa( 'd\'una imatge' )">
<a href="" onClick="avisa( 'd\'un vincle' );return false">Vincle buit</a>
...
```

3.- L'objecte Event

Per tal de manipular els events, JavaScript disposa d'una interfície, l'objecte **event**. Les seves propietats s'actualitzen de forma automàtica quan es produeix un event determinat, i les podem consultar per saber què ha passat, amb tota mena de detalls.

Si, en general, les diferències entre navegadors són importants, en aquest objecte hi ha un autèntic abisme. Com gairebé sempre, Internet Explorer ofereix més opcions i flexibilitat. A més, no totes les propietats es poden aplicar a tot tipus d'objectes. Si volem afinar,

haurem de consultar la documentació específica de cada fabricant per a cada objecte:

Propietat	Informació
x	Coordenada X del ratolí, referida a la finestra
y	Coordenada Y del ratolí, referida a la finestra
type	Cadena (String) que representa el nom de l'event

Les úniques propietats compatibles entre tots dos navegadors són **x**, **y** i **type**, de manera que, tot i que el tractament de l'objecte "event" proporciona una potència i control insospitats, pràcticament sempre haurem de detectar el navegador i oferir solucions diferents.

El plantejament del sistema d'events també és radicalment oposat entre els dos navegadors:

- Els navegadors del DOM 2 permeten que l'objecte que rep l'event el pugui processar i, si no, el passen a l'objecte pare de la jerarquia, seguin aquest ordre: vincle > contenidor > document > finestra > comportament per defecte. Pràcticament tots els objectes poden processar events.

- Netscape 4.7 fa tot el contrari, intenta processar l'event a la part més alta de la jerarquia (window) i, si no el pot manejar, el passa al seu inferior (document) fins arribar a l'objecte que l'ha generat. Això fa que pocs elements tinguin l'oportunitat de processar events.

Una forma, barroera però efectiva, de forçar Netscape 4.7 perquè processi els events, per exemple, en una capa, és escriure-hi un script intern que redefineixi el(s) comportament(s):

En aquest cas farem una capa que reaccioni als events **onmouseover** i **onmouseout** en tots dos navegadors. Després d'escriure-ho a l'etiqueta de la forma habitual, obrim un script i fem dues funcions amb el mateix nom dels events i el mateix resultat que hem definit a l'etiqueta.

```
<div style="position:absolute; left:100; top:80;"
onmouseover=" alert('Ratolí sobre la capa\nInternet Explorer'); "
onmouseoOut=" alert('Ratolí fora de la capa\nNavegador DOM 2'); ">
Aquí podem posar el<br>contingut de la capa
<script language="JavaScript" >
function onmouseover() {
  alert( 'Ratolí sobre la capa\nNetscape 4.7' );
}
function onmouseout() {
  alert( 'Ratolí fora de la capa\nNetscape 4.7' );
}
</script>
</div>
```

4.- Capturar events

Una manera més efectiva d'assegurar que els objectes respondran a uns determinats events és fer-ne una captura. És diferent per als dos navegadors. Amb Netscape 4.7 hem d'utilitzar el mètode **captureEvents()**, abans de començar. L'event es passarà com a paràmetre a la funció controladora. Explorer, en canvi, reconeix l'event dins de la funció.

Farem que la pàgina (document) capturi l'event MouseDown i el processarem amb una funció que ens dirà el tipus d'event i el lloc de la pàgina on s'ha produït. En primer lloc detectarem el navegador. A continuació, si és Netscape 4.7, fem que el document capturi l'event:

```
if ( ns ) document.captureEvents( Event.MOUSEDOWN )
```

Si volem capturar més d'un event els podem separar amb una barra vertical:

```
document.layer1.captureEvents( Event.CLICK | Event.MOUSEMOVE )
```

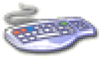
A continuació hem de fer una funció - comportament - que processi l'event:

```
function processa( e ) { ... }
```

Finalment, hem d'enregistrar la funció com a comportaments de l'objecte:

document.onmousedown = processa

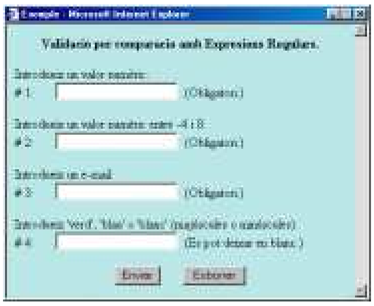
A dins de la funció, hem de detectar si no s'utilitza Netscape 4.7 per capturar l'event de la finestra. Internet Explorer no reconeix el paràmetre, per això escrivim: if (! ns) e = window.event . Finalment, amb el tipus d'objecte i les coordenades, fem un missatge d'alerta.



```
<script language="javascript">
var ns = document.layers
if ( ns ) document.captureEvents( Event.MOUSEDOWN )
function processa(e) {
  if( !ns ) e = window.event
  var txt = 'S\'ha capturat un event del tipus ' + e.type + '\n'
  txt += 'a la posició X: ' + e.x + ', Y: ' + e.y + '.\n'
  alert( txt )
  return true
}
document.onmousedown = processa
</script>
```

En aquesta pràctica tractarem de la validació de les dades introduïdes per l'usuari visitant de la nostra pàgina.

Hi ha ocasions que l'usuari ha d'introduir informació sol·licitada per la pàgina, per exemple dades d'un formulari que posteriorment s'ha d'enviar, valors numèrics per processar en un programa de càlculs, valors textuals acotats entre poques opcions de resposta a algunes preguntes. etc.



No podem refiar-nos que la informació introduïda per l'usuari estigui d'acord amb el format de les dades esperades. Introduir un text quan s'espera un número, una adreça de correu sense el símbol @, etc. solen ser errors habituals.

Per tant, el programa JavaScript ha de verificar que els valors introduïts siguin coherents, és a dir, ha de validar l'entrada d'informació, i si no és correcta, advertir l'usuari del seu error.

La manera de verificar si la informació és correcta es fa analitzant les dades introduïdes mitjançant comparacions amb els possibles valors correctes, o bé, comparant amb expressions regulars.

Elements del llenguatge en aquest capítol

Propietat **length** : Conté la longitud de la cadena en nombre de caràcters.
Mètode **charAt(n)** : Retorna el caràcter situat en la posició *n* dins la cadena, començant per 0.
Mètode **substring(inici,final)** : Retorna la subcadena ubicada entre les posicions *inici-final* d'una cadena.
Mètodes **indexOf(car)** i **lastIndexOf(car)** : Retornen la posició dins una cadena d'un caràcter *car*, començant a buscar des del principi o des del final, respectivament. Retornen -1 si no l'han trobat.
Mètodes **toLowerCase()** i **toUpperCase()** : Converteixen a minúscules i majúscules, respectivament, els caràcters d'una cadena.
Mètode **isNaN(valor)** : Retorna *true* si *valor* és valor no numèric (is not a number). En cas contrari, si *valor* és numèric, retorna *false*.
Mètode **match(patró)** : Retorna la cadena coincident amb el *patró* si l'ha trobat, sinó retorna *null*.
Metode **replace(patró, subst)** : Substitueix la cadena *subst* per la subcadena que coincideix amb el *patró*.
Expressions Regulars.

1.- Validació per comparació.

Dreamweaver incorpora una opció a "Ventana/Comportamientos/+/Validar formularios" que ofereix opcions de validació de dades de formularis. Les opcions són limitades i ho fa per comparació amb la incorporació de la funció genèrica MM_validateForm().

En aquest apartat farem una funció de nom **valida()**, que servirà per fer unes validacions per comparació, similars a les que introdueix el Dreamweaver, però, fetes per nosaltres. Així les entendrem millor i ens serà més fàcil adaptar-les a qualsevol altra situació.

Per fer l'anàlisi del text a validar és necessari treballar amb cadenes. Per això, ens seran molt útils els mètodes i propietats de l'objecte String. S'ha de tenir en compte que els caràcters que formen una cadena es numeren progressivament començant amb el índex 0 per al primer caràcter. Al principi d'aquesta pràctica hi ha una descripció dels mètodes que utilitzarem.

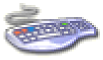
La següent funció **eliminaBlancsExtrems()** és una funció auxiliar que elimina els espais en blanc al principi i al final de la cadena que se li passa com a paràmetre i retorna la mateixa cadena sense els espais en blanc.

De vegades, sol passar que l'usuari introdueix de forma involuntària espais en blanc al principi i/o al final del text. Aquests espais en blanc solen ser difícils de localitzar ja que no es veuen, sobretot si estan al final. Podem ajudar a evitar aquests tipus d'errors implementant la funció que elimina aquests espais en blanc.

La línia 3 comprova si el primer caràcter (*charAt(0)*) és igual a un espai en blanc (' '). Si ho és s'extreu la subcadena situada entre la posició 1 i el final de la cadena (*cadena.length*) i l'assigna un altre cop a la variable **cadena**. Aquest procés es repeteix tantes vegades com calgui (*while*) mentre el primer caràcter de la cadena contingui un espai en blanc.

La línia 4 fa el mateix que la 3, però, referent als espais en blanc que hi pugui haver al final de la cadena. Per això comprova si hi ha un espai en blanc al final de la cadena tot comprovant la posició c*harAt(cadena.length-1)*. Recordem que *length* retorna la longitud de la cadena en nombre de caràcters, i que la cadena comença a comptar els caràcters amb el 0, per tant l'última posició tindrà l'índex ...*length -1*. Això ho repeteix mentre hi hagi espais en blanc al final (*while*).

Un cop s'han eliminat els espais en blanc al principi i al final, es retorna el valor resultant (línia 5). Aquesta funció s'invocarà des de dins la funció *valida()*.

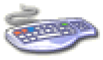


```
1 function eliminaBlancsExtrems(cadena)
2 {
3   while (cadena.charAt(0)==' ') cadena = cadena.substring(1,cadena.length);
4   while (cadena.charAt(cadena.length-1)==' ')
                                        cadena = cadena.substring(0,cadena.length-1);
5   return cadena;
6 }
```

A continuació, iniciarem la creació comentada de la funció de nom **valida()**, que serà l'encarregada de validar els diferents camps del formulari indicat al final d'aquest apartat. Repetirem el procés de validació per a cadascun dels quatre camps que conté el formulari. Si és correcte la variable **resCampX** valdrà *true*, sinó valdrà *false*. Mireu abans l'últim bloc de codi d'aquest apartat on hi ha el codi HTML amb els camps dins el formulari que volem validar.

El camp no. 1 és un camp que ha de contenir dades numèriques obligatòriament. A la línia 9 assignem a **valCamp1** el contingut del quadre de text *camp1*, eliminant-ne prèviament els possibles espais en blanc que s'hagin introduït al principi i al final amb la funció *eliminaBlancsExtrems()*.

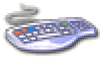
A la línia 9 assignem per defecte el valor *true* a la variable **resCamp1** i a la línia 11 verifiquem si el camp 1 és buit o conté un valor no numèric amb **isNaN()**. Si és així, assigna a *resCamp1* el valor *false*.



```
7   function valida()
8   {
9       var valCamp1=eliminaBlancsExtrems(document.form1.camp1.value);
10      var resCamp1=true;
11      if (valCamp1 =='' || isNaN(valCamp1)) resCamp1=false;
```

Les línies de codi següents corresponen a la validació del camp no. 2. Aquest Camp no. 2, també és obligatori i ha de contenir un valor numèric comprés entre -4 i 8. A les línies 12 i 13, assignem a **valCamp2** el valor de *camp2* sense espais en blanc extrems i a **resCamp2** el valor *true*, respectivament.

A la línia 14 és la que fa la validació pròpiament dita, i consisteix en efectuar comparacions, si **valCamp2** és una cadena buida, o és un valor no numèric o és inferior a -4 o és superior a 8, s'assigna a **resCamp2** el valor *false*.



```
12   var valCamp2=eliminaBlancsExtrems(document.form1.camp2.value);
13   var resCamp2=true;
14   if (valCamp2 =='' || isNaN(valCamp2) || valCamp2<-4 || valCamp2>8)
```

resCamp2=false;

A continuació s'efectua la validació de l'adreça electrònica sol·licitada com a obligatòria en el Camp no. 3. La majoria de validacions d'una adreça de correu solament verifiquen l'existència del símbol @ dins el text. Així és com ho fa el Dreamweaver. El codi següent efectua una validació més detallada.

Primer hem de tenir clar quina és l'estructura mínima d'una adreça e-mail, l'estructura general és: **x@y.dom** on **x** potser un o més caràcters alfabètics o numèrics, majúscules o minúscules, o un punt . o una ratlla baixa _ . **y** potser un o més caràcters alfabètics minúscules o numèrics i pot incloure el "." . **dom** és un dels dominis de primer nivell existent, per exemple: .es, .com, .fr, .org, .net, etc.

Lògicament, aquest codi solament verifica que l'estructura de l'e-mail sigui correcta, el que no pot fer és comprovar si aquest e-mail existeix realment.

La línia 15 llegeix el valor del camp de text del formulari i l'assigna a **valCamp3**, havent eliminat prèviament els espais en blanc inicials i finals si hi fossin. Les línies 16 i 17 assignen el valor inicial a **resCamp3** i verifiquen que l'adreça no sigui buida, respectivament.

A continuació de la línia 17, es fan múltiples comprovacions. Si alguna d'elles no és correcta, s'assignarà el valor *false* a **resCamp3**.

La línia 18, s'assegura que dins la cadena no hi hagi cap espai en blanc. La línia 19 cerca la posició d'un punt començant des del final i assigna aquesta posició a la variable **posUltimPunt**. A la línia 20, es verifica que aquest últim punt existeix.

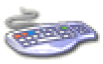
Sabem que des de la posició de l'últim punt fins el final de la cadena s'hi ha de trobar el nom del domini de primer nivell (.es, .org, .com, ...). Per tant, a la línia 21 assignem a la variable **domini** el contingut que es troba des de la posició de l'últim punt fins al final de la cadena, fent una extracció amb **substring** entre la posició de l'últim punt (*posUltimPunt*) i el final de la cadena determinat amb la propietat *length*.

La línia 22 verifica que el contingut de la variable **domini** correspongui amb algun dels dominis existents. En aquest exemple solament hem posat 5 dominis. Si desitgem ampliar-ho a tots els altres coneguts, s'hauran d'afegir dins la condició del *if* amb el següent format: **...&&domini!='.dom'**.

La línia 23 assigna a **longDomini** el valor de la longitud de **domini**, a la 24 determinem la posició del símbol @ i a la 25 comprovem que no existeixi més d'un símbol @ fent una nova cerca d'aquest símbol des del final de la cadena. Si solament existeix una @ les dues cerques de posició ens donaran el mateix resultat.

A la línia 26 verifiquem que la posició de @ no estigui situada com a primer caràcter ni tampoc com a últim caràcter tenint en compte la longitud del domini.

Si la validació de l'adreça e-mail passa correctament totes aquestes proves, **resCamp3** valdrà *true*, sinó valdrà *false*.



```
15   var valCamp3=eliminaBlancsExtrems(document.form1.camp3.value);
16   var resCamp3=true;
17   if (valCamp3 == '') resCamp3=false;
18   if (valCamp3.indexOf(' ')!=-1) resCamp3=false;
19   var posUltimPunt = valCamp3.lastIndexOf('.');
20   if (posUltimPunt==-1) resCamp3=false;
21   var domini=valCamp3.substring(posUltimPunt,valCamp3.length);
22   if (domini!='.es' && domini!='.com' && domini!='.net' && domini!='.org' &&
                                             domini!='.fr') resCamp3=false;
23   var longDomini=domini.length;
24   var posArroba=valCamp3.indexOf('@');
25   if (posArroba!=valCamp3.lastIndexOf('@')) resCamp3=false;
26   if (posArroba<=0 || posArroba>=valCamp3.length-longDomini-1) resCamp3=false;
```

Les línies següents corresponent a la validació del camp 4. Aquest camp solament pot tenir tres possibles valors, en majúscules o minúscules, o deixar-lo en blanc.

Com que és igual que estigui en majúscules o en minúscules, abans de fer les

comparacions, hem de convertir tots els caràcters de la cadena a minúscules amb la funció **toLowerCase()** de la línia 29. També podríem passar-ho tot a majúscules amb **toUpperCase()**, en aquest cas, les comparacions de la línia 30 s'haurien de fer amb el text en majúscules.

La línia 30 és on es fa la validació. És compara amb els tres valors possibles o bé amb una cadena buida, si és que és en blanc. Si no es compleix la condició **resCamp4** valdrà *false*, de cas contrari, valdrà *true*.

```
27   var valCamp4=eliminaBlancsExtrems(document.form1.camp4.value);
28   var resCamp4=true;
29   valCamp4=valCamp4.toLowerCase();
30   if (valCamp4!='verd' && valCamp4!='blau' && valCamp4!='blanc' && valCamp4!='')
                                                                     resCamp4=false;
```

En aquest punt de la funció **valida()** ja tenim quatre variables **resCampX** amb el valor *true* o *false*, segons hagi estat el resultat de la validació. Ara queda advertir a l'usuari dels possibles errors, si hi són.

A les línies 31 i 32, si totes les variables *resCampX* valen *true,* assignem a la variable **noError** el valor *true* de cas contrari, assignem *false*. És a dir, **noError** val *false* si hi ha alguna validació incorrecta.

La línia 33 conté un **alert** que mostra el resultat a l'usuari. Per a cada camp, mostrem "OK!" o" Error" mitjançant l'operador ternari **? :** , i un text indicador del resultat final avaluant la variable **noError**.

La línia 34 retorna el resultat global (indicat per **noError**) de la validació a l'event *onSubmit* que ha invocat a la funció *valida()*. Si es retorna *true*, s'efectuarà l'acció d'enviament de les dades, en canvi, si retorna *false*, quedarà anul·lada l'acció d'enviament de les dades i ens quedarà el formulari pendent per modificar els errors que hi hagi.

```
31   if (resCamp1 && resCamp2 && resCamp3 && resCamp4) var noError=true;
32   else var noError=false;
33   alert('Camp 1 : ' + ((resCamp1)?'OK!':'Error!') + '\n' +
             'Camp 2 : ' + ((resCamp2)?'OK!':'Error!') + '\n' +
             'Camp 3 : ' + ((resCamp3)?'OK!':'Error!') + '\n' +
             'Camp 4 : ' + ((resCamp4)?'OK!':'Error!') + '\n\n' +
             ((noError)?'Tot correcte, les dades s\'enviaran ara.'
             :'Tens errors, verifica\'ls abans d\'enviar.'));

34   return noError;
35   }
```

Finalment, a continuació, tenim el codi HTML on està inclòs el formulari d'exemple i les dues funcions descrites prèviament.

Com a element destacable podem observar com s'invoca l'event Submit mitjançant el controlador d'events **onSubmit**, dins la línia d'inici del formulari. Aquest event s'invocarà quan fem clic a sobre del botó "Enviar".

L'event Submit invoca a la funció **valida()** de la següent forma: **return valida();** . Això fa que el resultat de la validació (*true* o *false*) es passi a l'acció d'enviament del formulari, de tal manera que si el valor retornat és *true*, s'efectuarà l'acció d'enviament, en canvi, si és *false*, no s'efectuarà cap enviament.

```
<html>
<head>
<title>Exemple</title>

<script language='JavaScript'>
     ... Introduïm les línies 1 - 35 descrites prèviament ...
</script>

</head>
```

```
<body bgcolor='#BAE8E7'>
  <form name='form1' method='post' action='mailto:usuari@servidor.dom'
                                               onSubmit='return valida();'>
      Introdueix un valor numèric:<br>
      # 1: <input type='text' name='camp1'> (Obligatori.)<p>
      Introdueix un valor numèric entre -4 i 8:<br>
      # 2: <input type='text' name='camp2'> (Obligatori.)<p>
      Introdueix un e-mail:<br>
      # 3: <input type='text' name='camp3'> (Obligatori.)<p>
      Introdueix 'verd', 'blau' o 'blanc' (majúscules o minúscules):<br>
      # 4: <input type='text' name='camp4'> (Es pot deixar en blanc.)<p>
      <input type='submit' value='Enviar'> <input type='reset' value='Esborrar'>
  </form>
</body>
</html>
```

2.- Validació mitjançant comparacions amb "Expressions Regulars".

A l'apartat anterior, el codi es complica excessivament quan les comparacions a efectuar són molt variades. En aquest apartat introduirem el concepte de Expressions Regulars que ens permetran fer patrons de validació més complexos de forma més senzilla. Aplicarem aquests conceptes per substituir part del codi JavaScript descrit a l'apartat anterior.

Per entendre ràpidament el que són les Expressions Regulars recordarem els 'comodins' * i ? utilitzats a l'MS-DOS i Windows per seleccionar múltiples fitxers. Podríem dir que les Expressions Regulars són una evolució molt més avançada d'aquells rudimentaris 'comodins'.

El tema de les Expressions Regulars és molt extens, disposa de moltes opcions i la seva aplicació és comuna a altres llenguatges de programació diferents del JavaScript, com ara, el Perl, etc. En aquest apartat no tractarem aquest tema amb extensió. Solament descriurem aquells aspectes introductoris necessaris per aplicar-ho al codi de l'apartat anterior.

Podeu buscar més informació sobre el tema a Internet, llibres especialitzats o, per començar, veure l'ajuda de Dreamweaver (Ayuda/Referencia/JavaScript.... cercant la paraula RegExp).

Una Expressió Regular és un patró per a cadenes de caràcters. Tenim dues formes de crear els patrons corresponents a les Expressions Regulars. Mitjançant l'ús de l'objecte **RegExp** o bé declarant el patró com una expressió literal situada entre dues barres / (**/patró/**). Aquí utilitzarem la segona forma.

Un cop tenim creat el patró, el podem utilitzar en els mètodes *exec()* i *test()* de l'objecte RegExp o els mètodes *match()*, *replace()*, *search()* i *split()* de l'objecte String. Els principals caràcters especials per generar patrons són:

**** Indica que el següent caràcter és especial i no s'ha de considerar un literal.
^ Permet trobar coincidències que comencen al principi de la cadena.
$ Permet trobar coincidències que comencen al final de la cadena.
***** Permet trobar coincidències dels caràcters que el precedeixen 0 o més vegades.
+ Permet trobar una o més coincidències dels caràcters que el precedeixen.
? Permet trobar 0 o una coincidència dels caràcters que el precedeixen.
. Localitza qualsevol caràcter sempre que no estigui al principi de la cadena.
a|b Localitza qualsevol dels patrons (a o b).
[...] Localitza qualsevol dels caràcters situats dins els [ ].
\d Localitza dígits dins la cadena.

Disposem també dels modificadors següents:
g Aplica el patró de forma global a tota la cadena, no solament a la primera coincidència.
i Ignora si els caràcters són majúscula o minúscula.

Els modificadors s'introdueixen al final de la Expressió Regular, després de la / final.

Els següents exemples de patrons d'expressions regulars ens seran útils per aplicar-los en el codi de l'apartat anterior:

/[-+]?\d*\.?\d*/
Localitza números decimals.
[-+]? - Indica 0 o 1 caràcter + o - .
\d* - Indica 0 o més dígits numèrics.
\.? - Indica 0 o 1 punt.
\d* - Indica 0 o més dígits numèrics.

/[A-Za-z0-9._]+@[a-z0-9.]+(.es|.com|.net|.org|.fr)/g
Localitza cadenes amb el format adequat per ser considerades una adreça e-mail.
[A-Za-z0-9._]+ - Indica 1 o més caràcters entre A i Z, a i z, 0 i 9 o un punt o un guió baix.
@ - Indica el símbol literal @.
[a-z0-9.]+ - Indica 1 o més caràcters entre a i z, 0 i 9 o un punt.
(.es|.com|.net|.org|.fr) - Indica una de les cadenes de text separades amb | , si es desitja augmentar el nombre de dominis vàlids s'ha de fer afegint-los aquí.

/verd|blau|blanc|/gi
Localitza cadenes amb els continguts verd o blau o blanc o buit, en majúscules o minúscules. Fixeu-vos com després de la paraula blanc hem afegit un símbol | seguit de la finalització del patró /, aquesta part és la que permet seleccionar els camps buits. També s'ha inclòs el modificador g per indicar tota la cadena i per indicar majúscules i minúscules.

/^ +| +$/g
Localitza espais en blanc al principi o al final d'una cadena.
^ + - Cerca espais en blanc a l'inici de la cadena.
 +$ - Cerca espais en blanc al final de la cadena.

Utilitzarem també els següents mètodes:
- Mètode **match(patró)** : Retorna la cadena coincident amb el patró si l'ha trobat, sinó retorna null.
- Mètode **replace(patró, subst)** : Substitueix la cadena *subst* per la subcadena que coincideix amb el patró.

En el codi següent apliquem les Expressions Regulars al programa de validació vist a l'apartat anterior.

La línia que substitueix a les línies 3, 4 i 5 s'encarrega de substituir els espais en blanc del principi i final de la cadena. Per això cerca el patró d'un o més espais en blanc des del principi (^ +) o un o més espais en blanc des del final (+$) i els substitueix per una cadena buida ' '. El modificador *g* indica que es faci per a tota la cadena.

La part de codi valCamp1 != valCamp1.match(/[+-]?\d*\.?\d*/) que substitueix a la línia 11efectua una comparació entre el contingut de la variable *valCamp1* i el valor retornat per *match()* a l'aplicar el patró /[+-]?\d*\.?\d*/, si no coincideixen, vol dir que el contingut del *camp1* no és un número.

A la línia 14 es fa una acció similar a la indicada en el paràgraf anterior.

El codi valCamp3.match(/[A-Za-z0-9._]+@[a-z0-9.]+(.es|.com|.net|.org|.fr)/g) de la línia que substitueix a les línies 17-26 comprova el contingut de la variable *valCamp3*. Per això, s'extreu una cadena de la variable *valCamp3* segons el patró indicat com a paràmetre de *match()*. Si aquesta cadena no coincideix amb el valor que conté la variable *valCamp3*, vol dir que l'adreça e-mail no s'ajusta al patró.

El codi valCamp4!=valCamp4.match(/verd|blau|blanc|/gi) de la línia que substitueix a les línies 29 i 30 comprova si el valor retornat per *match()* segons el patró indicat, és diferent del contingut de *valCamp4. S*i és diferent vol dir que el contingut de *valCamp4* no s'ajusta als requeriments sol·licitats pel *camp4*.

Observem que la utilització de les expressions regulars redueix el codi necessari per fer validacions per comparació, sobretot si aquest codi de comparació és extens com en el cas de la validació de l'adreça e-mail.

En el cas concret de la validació de l'adreça e-mail, la utilització de les expressions regulars

ens permet ajustar-ho amb més precisió. Si en l'apartat anterior, s'acceptava qualsevol caràcter per formar l'e-mail en els texts previs i posteriors a l'@, en aquests cas, aquests caràcters són determinats dins els [ ] i es pot evitar la introducció de caràcters estranys.

Substituïm les següents línies pel codi indicat a continuació:

la 3, 4 i 5 per:

```
return cadena=cadena.replace(/^ +| +$/g,'');
```

la 11 per:

```
if (valCamp1 =='' || valCamp1 != valCamp1.match(/[+-]?\d*\.?\d*/)) resCamp1=false;
```

la 14 per:

```
if (valCamp2 =='' || valCamp2!=valCamp2.match(/[+-]?\d*\.?\d*/) || valCamp2<-4 ||
                                                        valCamp2>8) resCamp2=false;
```

de la 17 fins la 26 per:

```
if(valCamp3 != valCamp3.match(/[A-Za-z0-9._]+@[a-z0-9.]+(.es|.com|.net|.org|.fr)/g))
                                                                  resCamp3=false;
```

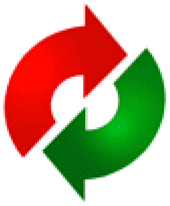
la 29 i 30 per:

```
if (valCamp4!=valCamp4.match(/verd|blau|blanc|/gi)) resCamp4=false;
```

Interacció i comunicació > La pàgina respon

Les pàgines estàtiques, amb una orientació purament informativa, van deixant lloc a d'altres, més intel·ligents, que es mouen seguint els interessos dels seus autors i dels usuaris. De l'aparador es passa a l'intercanvi d'informació.

Sota el títol "La pàgina respon" agrupem dos aspectes més relacionats amb aquest intercanvi que no pas amb el disseny: veurem com podem saber més del que els usuaris ens volen dir; aprendrem a protegir les pàgines de manera que només hi pugui entrar qui conegui la clau d'accés, desactivarem el botó dret, ...



En definitiva, no es tracta d'efectes espectaculars,sinó més aviat de conèixer unes tècniques que ens aportaran més control sobre el que passa amb les nostres pàgines.

Conceptes JavaScript en aquest capítol

Comportament **oncontextmenu :** acció quan s'activa el menú de context amb el botó dret.
Mètode **history.go() :** salta a la pàgina de l'historial especificada i al lloc on l'havia deixada.

1.- Camps ocults

En un formulari hi pot haver dades que s'enviaran i/o serviran per processar la resposta i que l'usuari no veu. Es tracta de camps que l'autor introdueix amb el modificador "hidden", ocult.

Aquest tipus de camps ens poden resultar molt útils en algunes ocasions. Per exemple, podem donar a un mateix formulari diverses utilitats depenent del valor d'un camp ocult: podem enviar un mateix formulari a llocs diferents, filtrar els missatges, obtenir informació sobre el client, ...

Els camps ocults s'utilitzen com qualsevol altre camp d'un formulari:

```
<input type="hidden" name="noEsVeu" value="ocult">
```

En aquest exemple prepararem un formulari que, a més del nom i el missatge de l'usuari, tindrà altres dades que ens poden ser útils: títol i adreça de la pàgina, navegador, versió, sistema, pantalla i el temps que ha trigat a carregar-se la pàgina.

Al cos farem un formulari amb 8 camps, sis d'ocults i dos de visibles. També hi posarem un botó per esborrar i un altre per enviar. S'enviarà pel mètode POST, sense encriptar (enctype="text/plain"), com a acció tindrà un "mailto:" a la nostra adreça i definirem onSubmit="return noDupli(this)", és a dir, que en el moment d'enviar-lo es passi a si mateix com a paràmetre d'una funció.

Assignarem l'event onLoad del cos a una altra funció que s'executarà quan s'acabi de carregar la pàgina. Aquesta funció, Fet(), posarà a la variable **fi** el temps actual.

A la capçalera declararem tres variables:
- **inici**: li assignem el temps actual quan comença la càrrega de la pàgina.
- **fi**: ja esmentada, servirà per calcular el temps de càrrega. No la inicialitzem.
- **trameses**: de moment a 0, evitarà que s'enviï el formulari dues vegades.

A continuació, la funció **noDupli()** rep com a paràmetre una referència al formulari. Comprova que el formulari existeix i, si és així, assigna als camps ocults les dades que volíem obtenir, entre elles, la resta (fi - inici) que ens diu quant ha trigat la pàgina a carregar-se. Després incrementa les trameses i, si encara són menys de 2 (és a dir, la primera) retorna a "true" i permet que s'enviï el formulari. En tots els altres casos retorna a "false".

Bé, ja tenim un formulari que ens informa més del que sembla i, de passada, hem vist com evitar trameses duplicades. Això només és el principi. Més endavant, quan tractarem les

Cookies, caldrà recordar que els seus valors, que poden haver estat dies guardats a l'ordinador del client, també es poden assignar a camps ocults. Això ens permetria, per exemple, fer un seguiment detallat de les visites de cada usuari.

A l'exemple, en lloc d'enviar el missatge per correu, hem posat un camp de text on es mostren els valors dels camps ocults que s'envien.

```
<script language="javascript">
var inici = new Date()
inici = inici.getTime()
var fi
var trameses = 0
function noDupli(f1) {
  if (f1) {
    f1.pag.value = document.title
    f1.tmp.value = (fi - inici) + ' mil·lisegons'
    f1.url.value = location.href
    f1.sis.value = navigator.appName
    f1.ver.value = navigator.appVersion
    f1.pan.value = screen.width + 'x' + screen.height + ', ' + screen.colorDepth + ' bits'
    trameses++
    if (trameses<2) return true
  }
  return false
}
function Fet() {
  fi = new Date()
  fi = fi.getTime()
}
</script>
</head>

<body onLoad="Fet()">
<form name="form1" method="POST" enctype="text/plain"
  action="mailto:unnom@pie.xtec.net?Subject=Info" onSubmit="return noDupli(this)">
<input type="hidden" name="pag" value="">
<input type="hidden" name="tmp" value="">
<input type="hidden" name="url" value="">
<input type="hidden" name="sis" value="">
<input type="hidden" name="ver" value="">
<input type="hidden" name="pan" value="">
Nom:   <input type="text" name="nom" size="40" maxlength="60"><br>
<textarea name="missatge" rows="7" cols="48"></textarea><br>
<input type="reset" name="Reset" value="Esborrar les dades">  
<input type="submit" name="Enviar" value="Enviar el missatge">
...
```

2.- Afegir informació al formulari

L'exemple anterior va una mica carregat de camps. La veritat és que no cal. Podem fer el mateix, fins i tot de forma més elegant, sense utilitzar camps ocults. Un formulari amb el nom, el missatge i els botons és suficient.

La funció que comprovava els duplicats - i omplia els camps ocults -, s'haurà d'encarregar d'afegir aquesta informació a continuació del missatge. Hem preparat un exemple que ho fa, però ens hem oblidat d'escriure la solució.

No té massa gràcia afegir unes dades al cos del missatge, perquè l'usuari el podrà veure i, potser, no li agradarà. Si el nostre objectiu és recollir el màxim d'informació sobre els usuaris d'una pàgina, la millor opció és posar-hi alguna cosa senzilla i gormanda, com ara una votació o una enquesta. Podem recollir les dades en un camp ocult.

El que es planteja - som en un mòdul avançat - és descobrir la forma de fer-ho. Un parell de pistes: es basa en l'exemple anterior i es fa un bon ús de l'operador "**+=**".

3.- Accés restringit

JavaScript, com a llenguatge que pot executar el client, no disposa d'opcions de seguretat ni d'autentificació d'usuaris. Qualsevol pàgina que es carregui es podrà investigar, si més no, a partir de la còpia que queda a la memòria cau.

Podem intentar que no es guardi amb un parell de directives <meta>: una que li diu que la pàgina expira a l'instant i l'altra que demana que no es guardi. Però la pàgina ja és al navegador, i hi ha moltes maneres d'interceptar-la. Si volem una autentificació segura hem d'utilitzar llenguatges que s'executin al servidor.

Si la pàgina que volem protegir no conté informació vital, o el nostre servidor no permet tenir executables, podem usar trucs que amaguin la contrasenya. En aquest cas farem una pàgina que demana la clau i no deixa accedir a la seva codificació.

Per això, hem de fer que l'arxiu s'obri en una finestra nova, sense botons ni menús, i que vagi directament a demanar la clau. Si l'usuari l'encerta, carrega la pàgina protegida a la finestra base (**opener**) i tanca la nova. Si no, presenta un missatge - que esborra el contingut de la pàgina - i prepara un temporitzador per tancar la finestra al cap de dos segons.

Amb aquest mètode, la versió que queda a la memòria cau si han fallat els 'metas' dels usuaris que no han encertat la clau i només conté el missatge "Clau incorrecta". Mirem el codi, perquè la funció és prou senzilla i s'explica tota sola. La clau és "dhtml".

```
<meta http-equiv="Expires" content="0">
<meta http-equiv="Pragma" content="no-cache">
<script language="javascript">
function clau() {
  var pwd = prompt('Clau d\'accés', '')
  if (pwd == 'dhtml') {
    opener.location.href = 'd108m7p4.htm'
    window.close()
  }
  else {
    document.write( '<h2>Clau incorrecta</h2>' )
    setTimeout( 'window.close()', 2000 )
  }
}
</script>
</head>

<body onload="clau()">
<font face="Arial" size="3"><br><br><br><br><br><br><br>
Escriviu la clau per accedir a la pràctica següent</font>
...
```

4.- Desactivar el botó dret

Amb el botó dret del ratolí els navegadors activen un menú de context. Hi ha opcions com ara "veure el codi de la pàgina", "Guardar imatge", "Copiar fons", "Crea una drecera", ... De vegades, no ens interessa gens que els usuaris tinguin aquestes possibilitats.

Per exemple, tenim una imatge original que no volem que es pugui guardar o no volem que es pugui veure el codi. Doncs, obrim una finestra sense menús i desactivem el botó dret. En aquest últim cas, els usuaris espavilats obriran una nova finestra (amb Control + N o, en navegadors moderns, + U) i el visualitzaran amb el menú. Més difícil ho tindran, en canvi, si posem la pàgina que volem protegir en un marc.

Hi ha diverses maneres de desactivar el botó dret del ratolí. Al segon exemple d'aquesta pàgina n'hem utilitzat un. Ara provarem un model diferent. Es tracta, en tots dos casos, d'interceptar l'event **onmousedown**, enviar-ho a una funció que comprovi, segons el navegador, amb quin botó s'ha fet clic, i que bloquegi la resposta.

L'exemple anterior es fa enviant una alerta. En aquest, més sofisticat, senzillament retornem fals. Això sí, interceptem també l'event **oncontextmenu**, assignem un valor fals al retorn (en lloc del menú de context) i, altre cop, retornem fals. Amb aquesta estratègia, ni funciona el botó dret, ni passa res, senzillament, la seva funcionalitat ha desaparegut.

```
<script language="javascript">
function bDret(e){
  if (document.all) if (event.button == 2) return false
  if (document.layers) if (e.which == 3) return false
}
function cMenu(){
  event.returnValue=false
  return false
}
if (document.layers) document.captureEvents(Event.MOUSEDOWN)
document.onmousedown=bDret
document.oncontextmenu=cMenu
</script>
```

5.- Diferències subtils

En diverses ocasions, al llarg del curs, hem utilitzat l'expressió **location.reload()** per tornar a carregar una pàgina. L'objecte history té el mètode go() que permet anar a la pàgina de l'historial que li especifiquem: **history.go(-1)** va a la pàgina anterior, **history.go(+1)** va a la següent, ... Aleshores, **history.go(0)** torna a carregar la pàgina actual.

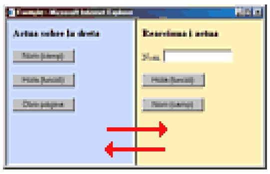
Pel que sembla, tenim dues maneres de fer el mateix: history.go(0) i location.reload(). Doncs no. La diferència està en l'objecte que utilitzem: mentre location guarda les adreces, history recorda el punt des d'on hem deixat una pàgina, com els botons del navegador.

La conseqüència és evident: si volem tornar a carregar la pàgina amb el cursor al principi, utilitzem location.reload(). Si, al contrari, volem anar a parar a un lloc precís de la pàgina, hem d'utilitzar history.go(0). Per exemple, si una pàgina té elements actius - vincles, formulari - que no es visualitzen a la primera pantalla (s'han d'usar les barres de desplaçament), i volem recarregar la pàgina a la mateixa posició, hem d'usar history.go(0).

Això és el que passa a l'índex d'aquest curs: com que és un llistat d'enllaços llarg, la majoria d'usuaris necessiten desplaçar la barra per accedir a les últimes posicions. Quan es fa clic en un vincle, l'índex es torna a carregar, destacant la pàgina actual. Si s'utilitzés location.reload() també funcionaria però, en aquests casos, el vincle actual quedaria per sota de l'àrea visible.

Si utilitzem només HTML, o fem les pàgines amb un editor (Dreamweaver), els marcs tenen més inconvenients que no pas avantatges. Són fàcils de crear, però el seu comportament és diferent segons el navegador. A més, és difícil mantenir-los sincronitzats i, és clar, al cap d'una estona de fer clic en vincles i tornar enrera, ja no sabem on som.

Aleshores, quan hem d'usar marcs? Doncs, quan els avantatges superin els inconvenients, o quan no hi hagi altre remei. Això sí, abans de començar hem de tenir clar que haurem de dedicar molta atenció a la sincronització.



Algunes de les eines que hem dissenyat per aquest curs es basen en una estructura de marcs. En tots els casos resultaven imprescindibles o, com a mínim, eren l'opció més efectiva i amb menys complicacions.

A l'**Inspector d'Objectes**, per exemple, el marc superior conté els controls, el de la dreta carrega la pàgina i el de l'esquerra l'analitza. La sincronització falla quan l'usuari segueix un vincle de la pàgina que s'analitza. El marc de l'inspector s'actualitza, però l'adreça de la pàgina - al quadre del marc superior, un control "file" - no canvia.

FerÍndex, en canvi, necessita un marc ocult per poder obrir les pàgines i extreure'n el títol. Ull viu amb aquesta tècnica: com que JavaScript només pot analitzar pàgines carregades en un navegador, amb aquest truc aconseguim veure-les i que no es noti.

Conceptes JavaScript en aquest capítol

Objecte **window.frames :** Array amb tots els marcs d'una finestra.
Propietat **frames.length :** número de marcs que conté una finestra.
Propietat **parent :** referència a la finestra que defineix una estructura de marcs.

1.- Crear l'estructura de marcs

Si es crea l'estructura de marcs en HTML és probable que les mides no siguin correctes en algun dels dos navegadors. Per tal d'evitar-ho, podem usar variables i codi JavaScript que ens facin les mides exactes per a cada navegador. Aquí proposem observar l'estructura que hem utilitzat a l'inspector d'objectes.

Un truc basat en l'estructura és el del frame zero: un marc ocult (es veu una ratlleta, si ens hi fixem bé) que s'utilitza com a finestra "visible per a la pàgina, invisible per a l'usuari". Es tracta de distribuir l'espai d'una finestra de manera que un dels marcs tingui dimensions 0. Ho hem d'utilitzar a l'eina **FerIndex** per tal de poder obrir les pàgines que analitza. No és l'única estratègia, però és pràctica, ràpida i neta, perquè gairebé no es nota.

Sempre que fem una estructura de marcs cal recordar que les versions 4 de Netscape no els actualitzen quan es redimensiona la finestra, o bé ho fan malament. Amb Dreamweaver es pot solucionar. Per això, anem al menú "Comandos" i seleccionem "Agregar/Quitar reparación de cambio de tamaño de Netscape ...". Podem aprofitar l'espai per a scripts que genera per inserir-hi les instruccions de correcció de mesures.

Al codi podem veure un bocí del que ha escrit Dreamweaver en lletra gris.

```
<script language="javascript">
<!--
function MM_reloadPage(init) {
  ...
}
MM_reloadPage(true);
var altura = (document.layers) ? 32 : 27;
document.writeln('<frameset rows="'+altura+',*" frameborder="0" framespacing="0">');
// -->
</script>
```

```
  <frame name="menu" src="menu.htm" scrolling="no">
  <frameset cols="280,*" frameborder="1" framespacing="1">
    <frame name="inspect" src="ajut.htm" scrolling="auto">
    <frame name="pag" src="portada.htm" scrolling="auto">
  </frameset>
</frameset>
<noframes>
  <body>
  <p>Aquesta pàgina utilitza marcs, i el vostre navegador no els pot mostrar.</p>
  </body>
</noframes>
```

2.- Referenciar marcs i objectes

En una estructura de marcs hi ha un document principal i diversos fills. Cada marc és una instància de l'objecte window, amb totes les seves propietats i mètodes. Normalment, l'arxiu on es defineix l'estructura no conté instruccions operatives, només conté les de creació dels marcs i, en tot cas, un bocí de codi sota l'etiqueta <noframes>.

Per tal de manipular els marcs, el DOM proporciona l'objecte **window.frames**, un Array on cada element és un objecte del tipus "window" que conté la pàgina d'un marc. Com tots els Arrays, podem referenciar els elements a partir del seu índex.

L'única propietat de l'objecte **frames** que no té window és **length**, que ens informa sobre el número de marcs continguts en una finestra. Amb aquest número, per exemple, podem fer un bucle que el vagi recorrent i ens informi sobre les característiques de cada marc.

Els marcs tenen un nom que s'assigna en el moment de la seva creació. Des de la finestra principal ens podem referir a un marc pel seu nom. Des de cada marc, ens podem referir a la finestra principal utilitzant la propietat **parent**. Si, des d'un marc, ens volem referir a objectes d'un altre, ho hem de fer a partir de "parent" i hem d'usar l'índex o el nom del marc:

```
parent.marc1.document.images[0].src = 'imatge1.gif'  … o bé …
parent.frames[0].dcocument.forms[0].nom.value = 'unNomQualsevol"
```

A continuació tenim un exemple que mostra diverses possibilitats. Crearem una estructura amb dos marcs, al 50%. Els anomenem "f1" i "f2". Els arxius que es carreguen també tenen aquests noms per simplificar-ho, però poden tenir-ne qualsevol altre. Un cop hem identificat els marcs pel seu nom, el dels arxius no té cap importància.

```
<html>
<title>Exemple</title>
<frameset cols="50%,50%">
  <frame name="f1" src="f1.htm">
  <frame name="f2" src="f2.htm">
</frameset>
</html>
```

Al marc de l'esquerra, arxiu **f1.htm**, farem un formulari amb tres botons, i els definirem l'event **onClick**. El primer botó carregarà la funció posaNom, que és en aquest mateix arxiu. El segon executarà la funció "hola()" en el marc f2, que encara no hem fet. El tercer tornarà a carregar una funció del seu propi arxiu, "obrePag()".

La funció **posaNom()** rep un paràmetre, txt, i l'assigna a un camp de text en un formulari de l'altre marc, que encara no hem fet. Per referenciar l'objecte utilitza la sintaxi d'índex:

```
parent.frames[1].document.forms[0].nom.value = txt
```

Hem dit que l'últim botó executa una funció, **obrePag()**. Rep com a paràmetre una adreça, url, i la utilitza per actualitzar el marc de la dreta. Utilitza la sintaxi del nom:

```
parent.f2.location.replace( url )
```

Després, prepara un temporitzador que tornarà a carregar, també a la dreta, l'arxiu original, f2.htm. Com podem veure, tots tres botons acaben actuant sobre el marc de la dreta, a través de funcions pròpies (botons 1 i 3) o, directament, "cridant" una funció de l'altre marc (botó 2). Això només és possible, evidentment, si sabem amb seguretat quina pàgina contindrà.

```
<html>
<head>
<script language="javascript">
function posaNom(txt){
  parent.frames[1].document.forms[0].nom.value = txt
}
function obrePag(url){
  parent.f2.location.replace( url )
  tm = setTimeout( 'parent.f2.location.replace("f2.htm")', 2000 )
}
</script>
</head>
<body bgcolor="#CCDDFF">
<form name="Form1">
<h3>Actua sobre la dreta</h3>
<input type="button" value="Nom (camp)" onClick="posaNom('Un Nom')"><br>
<input type="button" value="Hola (funció)" onClick="parent.f2.hola('esquerra')"><br>
<input type="button" value="Obrir pàgina" onClick="obrePag('prova.htm')">
</html>
```

Bé, només ens queda fer l'arxiu **f2.htm**, que es mostrarà al marc de la dreta. Aquest ha de tenir, de moment, un camp i una funció condicionats per l'arxiu anterior.

Al cos hi posarem un altre formulari amb tres controls. El primer serà el camp "nom" que, com hem vist, s'ha de poder actualitzar des de l'esquerra. El segon serà un botó que engegui la mateixa funció "hola()" que s'engegava des de l'altre marc. Ara, però, és al mateix arxiu i, per tant, s'ha de cridar d'una altra manera ... o no? Doncs, com vulguem, si ho escrivim com a l'altre, parent.f2.hola(), també funcionarà. La conclusió és evident.

L'últim control del formulari serà un altre botó, que fa una feina, si més no, curiosa: executa una funció de l'altre marc. El que té d'especial és que la funció actua sobre el marc de la dreta, que és el mateix que conté el botó. De manera que un control de la dreta li demana a una funció de l'esquerra que actualitzi un altre control de la dreta. Com ho veiem?

Finalment, la funció "**hola()**", que s'utilitza des de tots dos marcs, només fa una alerta a partir del text que rep com a paràmetre. A hores d'ara ja queda clar que els controls de formulari i les funcions són accessibles des de tots els marcs. També ho són les variables i tots els altres objectes, si utilitzem la referència correcta.

```
<html>
<head>
<script language="javascript">
function hola(txt){
  alert( 'Hola!\nEt saluda el marc: ' + txt )
}
</script>
</head>
<body bgcolor="#FFEEBB">
<form name="Form2">
<h3>Reacciona i actua</h3>
Nom: <input type="text" name="nom"><br>
<input type="button" value="Hola (funció)" onClick="hola('dreta')"><br>
<input type="button" value="Nom (camp)" onClick="parent.f1.posaNom('Un Altre')">
</html>
```

Interacció i comunicació > **Projecte interactiu**

Un cop hem tractat els events i els comportaments, hem vist la validació de dades dels formularis i hem provat la interacció a la pròpia pàgina i entre marcs, ens podem plantejar un tipus de pàgines per al projecte que no siguin merament informatives, sinó que "facin coses" en resposta a les accions dels usuaris i demostrin dominar la comunicació.

Preparem la pàgina "index.htm" de la carpeta "m7" de manera que contingui un botó o un vincle per obrir una nova finestra. La pàgina índex no anirà al projecte, però el contingut de la finestra sí. Podem fer que s'obri directament des d'una pàgina del projecte, si ho preferim. En qualsevol cas, el resultat d'aquest exercici ha d'anar en una finestra creada des de JavaScript, sense barres de menú ni botons ni adreça.

A la finestra hi ha d'haver un formulari que contingui, com a mínim, els camps: nom, adreça de correu i un quadre de text per a opinions. També els dos botons típics, per esborrar les dades i enviar el formulari. Hem de fer una funció de validació que comprovi que cap d'aquests tres camps queda buit i que l'adreça de correu té un format correcte.

El formulari ha d'enviar tota la informació possible sobre el
navegador, l'entorn i el temps de càrrega en un únic camp ocult.
Aquesta és una variació dels exercicis 1 i 2 de la pràctica 3. També hem de desactivar el botó dret del ratolí per tal de dificultar la visió del codi font.

Com sempre, cal comprovar que tot funcioni correctament amb diversos navegadors.

És una web o és un programa?

Una de les tendències amb més èxit és aquella que intenta que les pàgines web siguin cada cop més útils i pràctiques, sense que això vulgui dir, necessàriament, que s'hagi d'oblidar l'aspecte atractiu. I és que la gent navega a la recerca d'alguna cosa que l'interessa, i vol trobar-ho ràpid i sense problemes.

La "usabilitat" d'una web - tècniques que faciliten la feina - implica un disseny senzill i, sobretot, comprensible a primera vista. Si els usuaris veuen que necessitaran perdre temps per entendre'n la interfície, buscaran un lloc que els resulti més familiar.

Des d'aquesta perspectiva, una web és un entorn on l'usuari treballa, i hem de fer que sigui còmoda i eficaç, si volem millorar el seu grau de satisfacció. Com un programa. Quins són els que utilitzem habitualment, els de disseny espectacular o els que ens resolen els problemes?

El disseny, diuen, ha de ser com l'àrbitre en els esports: com menys es faci notar, millor.

Envieu un missatge a la vostra tutoria un cop l'exercici estigui penjat a Internet. Indiqueu sempre l'adreça de forma que s'hi pugui accedir directament.

Calaix desastre

Aquest mòdul fa un recull d'aquells temes que no han tingut una ubicació en els mòduls anteriors, però, que també són prou interessants de conèixer per la seva utilitat.

De ben segur que ens deixem moltes altres coses sobre DHTML i JavaScript, aquí, però, en fem un petit recull.

La **pràctica 1** introdueix les galetes, fragments d'informació que la pàgina desa en el nostre ordinador. La seva utilitat és important ja que permeten personalitzar una pàgina a les preferències de l'usuari.

Segur que en el vostre ordinador hi teniu desada alguna galeta. Mireu dins de la carpeta 'windows/cookies' si feu anar l'Internet Explorer, o mireu el fitxer cookies.txt que hi ha al directori d'instal·lació del Netscape, si feu anar el Netscape.

La **pràctica 2** mostra com es poden comunicar dues pàgines web mitjançant el pas de paràmetres introduïts en l'adreça URL. Aquests paràmetres serviran per crear interactivitat entre elles.

La **pràctica 3** és una introducció a la metodologia a utilitzar quan es volen crear aplicacions JavaScript complexes o programes extensos considerats com a aplicacions completes de gestió, control, etc. (amb les limitacions pròpies del fet d'executar-se en un navegador de client) i tenir les tècniques de depuració disponibles per corregir errors de sintaxi i disseny.

Optimitzar la feina feta és el principal objectiu de la **pràctica 4**. El temps invertit per fer una aplicació vàlida per a una pàgina ha de ser utilitzable per a d'altres pàgines que realitzem en el futur, són les llibreries de codi JavaScript i fulls d'estils reutilitzables. En aquest cas, el millor és poder-ho desar en fitxers independents de l'aplicació que fem i carregar-los quan sigui necessari.

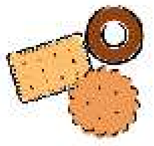
En aquest mòdul:

Pràctica 1: **Persistència de dades**
Pràctica 2: **Passar paràmetres**
Pràctica 3: **Fem aplicacions**
Pràctica 4: **Reutilitzar el codi**

Exercicis: **Pàgines amb memòria**

Per fer les pràctiques, obrim el Dreamweaver i ens situem a la "Vista de código", al mig de la secció <body> que, normalment, és a la línia 8. Allí hem d'inserir el codi dels exercicis. Si no tenim activada la numeració de línies, anem al menú *Ver - Opciones de vista de Código - Números de líneas* i l'activem.

Alguna vegada heu estat visitant una pàgina web que us ha reconegut i s'ha recordat de les vostres preferències, manifestades en alguna visita anterior? Per Internet es troben pàgines que tenen aquesta habilitat.



Però, com s'ho fan per recordar aquesta informació? La resposta està en les cookies, altrament dit, galetes.

Quin nom més curiós! Igual que altres conceptes d'Internet, aquest nom no té cap relació amb el que representa, però, ha estat acceptat per tothom.

Elements del llenguatge en aquest capítol

Propietat **document.cookie** : Facilita el treball amb les cookies.
Objecte **Date** : Objecte JavaScript per treballar amb dates.
Mètode **toGMTString()** : Retorna una cadena amb forma de data
Mètode **setTime()** : Estableix la data i hora d'un objecte Date.
Mètode **getTime()** : Retorna la data i hora d'un objecte Date.
Funcions **escape()** i **unescape()** : Codifiquen i descodifiquen els caràcters no ASCII.
Funció **replace()** : Substitueix un tros de cadena de text per un altre.
Funció **split()** : Descompon una cadena en un vector separant-la pel caràcter separador.

1.- Concepte i creació de cookies.

Les cookies ens serveixen per memoritzar dades d'una pàgina web entre diferents sessions de treball amb el navegador.

Es poden definir de dues maneres diferents: Mitjançant una crida al servidor HTTP i mitjançant JavaScript. En el primer cas, es fa utilitzant la directiva **Set-cookie:** situada a la capçalera de resposta HTTP originada per un programa CGI que s'executa al servidor. L'altre mètode consisteix en la definició local de la cookie mitjançant JavaScript.

En aquesta descripció no es tractarà del cas de cookies definides en el servidor HTTP, ja que la seva implementació requereix la creació de programes CGI situats en el servidor, i això, no és l'objectiu d'aquest curs. En canvi, tot el que tractarem a partir d'ara correspondrà al treball amb cookies mitjançant JavaScript.

Una cookie consisteix en un parell **nom=valor** que es pot desar en l'ordinador client.
IE desa aquests valors dins la carpeta "cookies" i NN ho fa dins el fitxer "cookies.txt".

La forma de fer-ho és mitjançant la propietat **document.cookie**. La seva sintaxi general és la següent:

```
document.cookie = "nom=valor;expires=data;path=cami;domain=domini;secure";
```

nom : Aquest és el nom de la cookie. A aquest paràmetre se li assigna el **valor** de la cookie. És l'única dada imprescindible per crear la cookie.

expires : Aquest valor estableix la data en format GMT (Greenwich Mean Time) a partir de la qual la cookie deixarà de tenir validesa. Si no s'especifica, es pren el valor de la data i hora actual, per tant la cookie caducarà en tancar el navegador. Habitualment s'acostuma a expressar la data corresponent a un nombre de dies especificat en crear la cookie.

path : Aquest valor estableix una expressió patró que representa el camí on tindrà validesa la cookie. Si no s'especifica, agafarà per defecte el camí de la pàgina web que ha creat la cookie.

domain : Aquest valor estableix una expressió patró que representa el domini en què tindrà validesa la cookie. Si no s'especifica agafarà per defecte el domini de la pàgina web que crea la cookie.

secure : L'existència d'aquest paràmetre estableix que només es pot enviar la cookie si s'ha accedit a un servidor segur per mitja del protocol HTTPS.

Exemples de creació de cookies:

- document.cookie = "Usuari=Joan Costa";

Es crea una cookie de nom "Usuari" i valor "Joan Costa". Com que no s'ha especificat cap altre valor, aquesta tindrà validesa mentre no es tanqui el navegador i dins el domini i path propis de la pàgina web que l'ha creat.

- document.cookie= "Color = blue; expires=' Thu, 11-Aug-2002 12:30:00 GMT ' ";

Cookie de nom Color i valor blue, que tindrà validesa pel domini i path de la pàgina web que l'ha creat, mentre no s'hagi superat la data del 11-8-2002 a l'hora especificada.

Les cookies també tenen en consideració l'**usuari identificat** en la sessió de Windows actual. Si aquest usuari no coincideix amb el que ha creat la cookie, aquesta no tindrà cap efecte, encara que coincideixin els altres paràmetres.

S'ha de tenir en compte que la cookie desa per defecte el camí i domini de la pàgina web que l'ha creat, això vol dir que qualsevol altra pàgina que s'hagi descarregat des del mateix camí i domini també podrà tenir accés al contingut de la cookie i, per tant, llegir i escriure el seu contingut. Això pot ser molt útil per **passar dades** d'una pàgina web a una altra. Si es desitja ampliar l'àmbit de pàgines on la cookie serà accessible, s'haurà d'introduir el contingut corresponent als valors **domain** i **path**.

Sempre que desitgem llegir el valor d'una cookie, s'haurà d'utilitzar la propietat **document.cookie**, sense indicar cap assignació. Aquesta ens retornarà una cadena formada pel conjunt de parelles **nom=valor** separades amb un **;** de totes les cookies que compleixen la condició de domini, path i data d'expiració del document que la sol·licita. La cadena retornada s'haurà de tractar per obtenir els valors individuals.

Si es desitja **esborrar una cookie** s'ha de fer posant-li una data de caducitat anterior a la data actual.

Tal com s'ha dit, és necessari treballar amb dates per establir la data de caducitat. Per això hem de conèixer bé el treball amb l'objecte Date. Si teniu algun dubte podeu consultar la pràctica 1 del mòdul 5. La data de caducitat s'ha d'establir en format GMT, això és fàcil d'implementar mitjançant el mètode **toGMTString()** de l'objecte Date que retorna una cadena de text amb aquest format.

Per exemple, si desitgem obtenir la data en format GMT, corresponent a 3 dies, a partir d'avui ho podem fer seguint els tres passos següents:

1 - Obtenim la data actual. *data* conté la data del sistema:

```
data = new Date();
```

2 - Establim la data al valor de la data actual més tres dies. Recordem que l'objecte Date emmagatzema la data en forma de mil·lisegons, per tant tres dies són: 3 dies * 24 hores/dia * 60 minuts/hora * 60 segons/minut * 1000 ms/s .

```
data.setDate( data.getDate + 3 *24*60*60*1000);
```

3.- Obtenim la data en format GMT:

```
data.toGMTString();
```

Amb això obtindrem la cadena de text amb el format GMT següent:

```
DiaSem, DD-Mes-YY HH:MM:SS GMT.
```

El següent exemple ens mostra la creació i eliminació de les cookie's. S'ha implementat mitjançant la creació de la funció **crearCookie()** perquè és el mètode més habitual i ens servirà per a exemples posteriors.

S'ha eliminat la utilització dels paràmetres *domain*, *path* i *secure*, per simplificar l'exemple i perquè són d'ús més específic i poc habitual, encara que la funció és fàcilment modificable per afegir aquests paràmetres si és necessari.

Si introduïm uns Dies de vida negatius, la cookie serà esborrada. Si no s'indica cap Dia de vida, la cookie solament existirà fins que es tanquni totes les finestres del navegador.

Si s'indica un determinat nombre de Dies de vida positius, la cookie tindrà vida durant

aquests dies fins la seva caducitat. En aquest cas la cookie queda desada dins la carpeta "cookies" en el IE o dins el fitxer "cookies.txt" en el NN. Si no s'indica cap data, la cookie queda emmagatzemada a la memòria i desapareix en tancar totes les finestres del navegador.

Observeu també, com s'utilitza la funció **escape()** per codificar aquells caràcters que no són ASCII, com per exemple, espais en blanc, accents, caràcters especials, etc. Aquesta funció és necessària per codificar els caràcters especials dins el valor de la cookie. Posteriorment, en llegir la cookie, haurem de fer el procés invers amb **unescape()**.

La funció **crearCookie()** també la utilitzarem en els següents apartats.

Dins el bloc <head>:

```
<script language='JavaScript'>
  function crearCookie(pNom, pValor, pDiesCaducitat)
  {
    if (pNom != ' ') {
      var txt = pNom + '=' + escape(pValor);
      if (pDiesCaducitat != ' ') {
        var dataCad = new Date();
        dataCad.setTime(dataCad.getTime() + pDiesCaducitat*24*60*60*1000);
        txt += '; expires=' + dataCad.toGMTString();
      }
      document.cookie = txt;
    }
  }
</script>
```

Dins el bloc <body>:

```
...
<form name="form1">
  Nom cookie: <input type="text" name="nom"><br>
  Valor: <input type="text" name="valor"><br>
  Dies de vida: <input type="text" name="dies">
  <input type="button" value="Crear Cookie" onClick="crearCookie(form1.nom.value,
                                              form1.valor.value, form1.dies.value);"><p>
  <input type="button" value="Veure les Cookie per aquest document"
                                                     onClick="alert(document.cookie);">
</form>
...
```

2.- Recollir les dades de les cookies.

Si una pàgina carregada en el navegador té cookies a la seva disposició, les pot obtenir mitjançant la propietat **document.cookie**. Aquesta ens retorna una cadena de text amb els parells **nom=valor** de les cookies.

A l'exemple anterior, fent clic sobre el botó "Veure les cookies ....", hem vist quina forma té la cadena que ens retorna. Aquesta està formada per un conjunt de parells **nom=valor** units per un punt i coma més un espai, de forma similar a la següent:

nom1=valor1; nom2=valor2; nom3=valor3; …….

Si desitgem obtenir el valor d'una sola d'aquestes cookies haurem de tractar la cadena, és a dir, descompondre-la en els diferents parells **nom=valor** i posteriorment verificar si existeix el nom que busquem i obtenir el seu valor.

En aquest apartat veurem com analitzem aquesta cadena per obtenir el valor desitjat d'una sola cookie.

Crearem una funció de nom **llegirCookie().** Passant-li com a paràmetre el nom de la cookie, ens ha de retornar el seu valor, en el cas que la cookie existeixi, o bé, **null** en el cas que no existeixi.

També aprofitarem aquest apartat per crear una altra funció més senzilla, però que també serà útil posteriorment: és la funció **esborraCookie()**. Passant-li el nom de la cookie, ens l'esborrarà assignant-li la data corresponent al 1-1-1970.

La funció **llegirCookie()** rep com a paràmetre el nom de la cookie a llegir. Si el nom del paràmetre **pNom** té un valor, s'assigna a la variable **dc** el contingut de *document.cookie*. Si *dc* té una longitud superior a 0, es descompon aquesta variable en un vector mitjançant **split()**. Aquesta descomposició crea un vector, que també s'assigna a *dc*, on cada element conté un parell *nom=valor*, ja que s'ha escollit com a element separador el ;.

Posteriorment, dins el bloc **while(...)**, s'analitza cadascun dels elements del vector *dc*. Cada element del vector *dc* es torna a descompondre mitjançant *split()*, assignant el vector resultant a la variable **parell**, el caràcter separador, en aquest cas és "=".

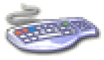
El vector **parell** constarà de dos elements, el *nom* i el *valor,* referits amb l'índex del vector 0 i 1 respectivament. Els noms corresponents a un índex **i** superior a zero, tindran afegit un espai en blanc al davant, que prové de la cadena retornada per *document.cookie.* Aquest espai en blanc afegit s'elimina amb la funció de cadena *replace()*.

La funció **replace()** cerca un patró dins la cadena i si el troba el substitueix per una **cadena nul·la** amb ''. El patró és: **/^ /** . Aquest indica que cerqui un sol espai en blanc a partir del principi de la cadena. Si el troba, serà substituït per la cadena nul·la.

Quan **parell[0]** ja conté correctament el nom de la cookie, aquest es compara amb el paràmetre *pNom*. Si coincideixen s'assigna a la variable **valor** el contingut de **parell[1],**prèviament es fa el **unescape()** per desfer la substitució de caràcters que no són ASCII.

Finalment, en acabar el recorregut pel vector **dc** dins el bloc **while(...)...** es retorna al contingut de la variable **valor**. Aquesta valdrà **null** si no s'ha trobat la cookie o valdrà el valor de la cookie si ha estat trobada.

La funció **esborraCookie()** fa el mateix que fèiem en l'apartat anterior quan volíem esborrar una cookie. A la cookie indicada pel paràmetre *pNom*, li assigna un valor de data, al paràmetre **expires** de la cookie, inferior a la data actual. En aquest cas li assigna la data 1-1-1970 determinat per dataCad=new Date(0), on s'ha utilitzat el constructor de Date amb un valor inicial de 0 mil·lisegons.



Funcions a afegir dins el paràmetre <script> del bloc <head>:

```
<script language="javascript">
function llegirCookie(pNom)
{
  var valor=null;
  var parell;
  if (pNom!='') {
    var dc=document.cookie;
    if (dc.length > 0) {
      dc=dc.split(';');
      var i=0;
      while (i<dc.length) {
        parell=dc[i].split('=');
        if (i!=0) parell[0]=parell[0].replace(/^ /,'');//elimina l'espai en blanc del principi.
        ++i;
        if (parell[0]==pNom) valor=unescape(parell[1]);
      }
    }
  }
  return valor;
}

function esborraCookie(pNom)
{
  var txt= pNom + '= null';
  var dataCad = new Date(0);
  txt += '; expires=' + dataCad.toGMTString();
  document.cookie = txt;
}
</script>
```

Afegir dins el paràmetre <form> del bloc <body>:

```
<input type='button' value='Llegir Cookie' onClick='llegirCookie(Nom de la cookie)'>
```

```
<input type='button' value='Esborrar Cookie' onClick='esborraCookie(Nom de la cookie)'>
```

3.- Exemple d'aplicació de cookies.

En aquest apartat farem una aplicació d'ús de les cookies. Consistirà en crear una pàgina web, que inicialment, sol·liciti a l'usuari el seu nom i l'idioma preferit. Posteriorment mostrarà un missatge personalitzat en aquest idioma. A més, també mostrarà el nombre de vegades que hem visualitzat la pàgina i la data de l'últim accés. La caducitat de les cookies serà de 30 dies.

Disposarem, també, d'un botó auxiliar per esborrar totes les cookies i reiniciar l'exemple sense haver d'esperar que caduquin les cookies.

Utilitzarem dos blocs amb codi JavaScript, dins la secció <body> i dins la secció <head>. El codi del bloc <head> conté les tres funcions descrites als apartats anteriors més una nova funció de nom *inicialitza()*. Aquestes funcions seran invocades des del codi JavaScript del bloc <body>.

La funció **inicialitza()** (línies 5-12) serveix per inicialitzar les cookies en el cas que en carregar la pàgina, aquestes no existeixin. Les cookies a inicialitzar són 4: **nom**, **idioma**, **comptador** i **dataUltimAcces**. Les dues primers agafen el valor dels camps *nom* i *idioma* del formulari *form1* que es crea des d'el bloc <body>. Un cop inicialitzades les cookies, es recarrega la pàgina amb la funció **reload()** de la línia 11.

Dins el bloc <body> tenim el codi JavaScript entre les línies 14-60, aquest codi s'executa immediatament després de carregar la pàgina. El primer que fa és obtenir el valor de cadascuna de les quatre cookies (línies 15-18). Si alguna d'elles no existeix, retornarà **null** a la variable **nomUsuari**, **idioma**, **comptador** o **dataUltimAcces** corresponent. En cas contrari, aquestes variables contindran el valor de la cookie.

A la línia 19 es verifica si cookie *nom* ha retornat **null**. Si és així, s'executarà el *codi d'inicialització* entre les línies 20-27. En cas contrari, s'executa el *codi de representació* (línies 29-52) del missatge personalitzat de la pàgina i el *codi d'actualització* de les cookies (línies 53-58).

El **codi d'inicialització** (línies 20-27) s'encarrega de treure per la finestra un text informatiu més un formulari que conviden a introduir el nom i les preferències respecte l'idioma. Un cop introduïdes aquestes dues dades, en prémer sobre el botó "OK!" s'invoca la funció **inicialitzacio()** i es creen les cookies amb els valors inicials.

El **codi de representació** del missatge (línies 29-52) és el que es mostra de forma habitual en la majoria dels casos. En funció del valor de la variable idioma s'executa un sol dels tres blocs que el composen, línies 29 -36 per al català, línies 37-44 per a l'anglès, i línies 45-52 per al francès. Conjuntament amb el text, es mostra el nom de l'usuari, el nombre de visites personalitzades i la data i hora de l'última visita, valors obtinguts de les variables *nomUsuari*, *comptador* i *dataUltimaVisita*.

Un cop s'ha mostrat el text informatiu personalitzat, les línies 53-58 s'encarreguen d'**actualitzar les cookies**. És necessari actualitzar aquests valors perquè com a mínim s'actualitzi la data de caducitat de les cookies. Aquest procés es fa rescrivint el valor de les cookies amb **crearCookie().**

Per a les cookies **nom** i **idioma** (línies 53 i 54) es rescriu el mateix valor, amb el qual solament s'ha actualitzat la data de caducitat. En el cas de la cookie **comptador** (línia 55), s'hi rescriu el valor de la variable *comptador* incrementat en una unitat, així, la propera visita es veurà el comptador incrementat.

Per a la cookie **dataUltimAcces** és necessari determinar prèviament el valor de la data actual i obtenir la cadena de data i hora en el format apropiat (línies 56 i 57), el resultat es desa a la variable **dataHora**. Per entendre les funcions de tractament de dates i hores podeu consultar la pràctica 1 del mòdul 5 on es detallen específicament .

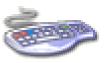
Un cop *dataHora* ja conté el valor apropiat, es rescriu la cookie *dataUltimAcces* a la línia 58.

Dins el bloc <body> i fora del bloc de codi JavaScript, s'introdueix la resta de codi HTML. En aquest cas solament hi ha definit un formulari que conté un botó per **esborrar les**

cookies. Aquest és un botó d'ús auxiliar, serveix per esborrar totes les cookies amb la funció **esborraCookie()** invocada per a cada cookie. D'aquesta forma, en tornar a obrir la pàgina d'exemple, ens apareixerà el formulari d'inicialització i podrem fer proves sense haver d'esperar que caduquin les cookies.

Per observar el correcte funcionament de l'exemple, s'ha d'obrir i tancar l'exemple múltiples vegades, observant com s'incrementa el comptador, es mantenen la data, les preferències d'idioma i nom d'usuari. Les proves s'han de fer tenint la finestra del navegador oberta o tancant totes les finestres del navegador, o tancant i tornant a obrir l'ordinador. S'ha de tornar a comprovar si funcionen al cap d'uns quants dies.

El botó "Esborrar totes les....." solament s'ha de polsar quan es desitgin reiniciar manualment totes les cookies sense esperar la seva caducitat. Això és per facilitar la realització de proves de funcionament.



Dins el bloc <head>:

```
<script language='JavaScript'>
 ... Introduïu la funció crearCookie() ...
 ... Introduïu la funció llegirCookie() ...
 ... Introduïu la funció esborraCookie() ...

 function inicialitzacio()
 {
   crearCookie('nom', document.form1.nom.value, '30');
   crearCookie('idioma',
     document.form1.idioma.options[document.form1.idioma.selectedIndex].
                                                              value, '30');
   crearCookie('comptador','1','30');
   crearCookie('dataUltimAcces',' ' ,'30');
   location.reload();
 }
</script>
```

Dins el bloc <body> :

```
 ...
<script language="JavaScript">
  var nomUsuari=llegirCookie('nom');
  var idioma=llegirCookie('idioma');
  var comptador=llegirCookie('comptador');
  var dataUltimAcces=llegirCookie('dataUltimAcces');

  if (nomUsuari==null || nomUsuari=='null') {
    document.writeln('Hola, sembla ser que és el primer cop que accedeixes a aquesta
                                                              pàgina<br>');
    document.writeln('Has d'omplir les següents dades per personalitzar-la.<p>');
    document.writeln('<form name=\'form1\'>');
    document.writeln('Introdueix el teu nom: <input type=\'text\' name=\'nom\'>
                                                              <p>');
    document.writeln('Selecciona idioma:     
              <select name=\'idioma\'>
              <option selected value=\'ca\'>Català<option value=\'an\'>Anglès
              <option value=\'fr\'>Francés</select>');
    document.writeln('<input type=\'button\' value=\'  OK!  \'
               onClick=\'inicialitzacio();\'>');
    document.writeln('</form>');
  }
  else {
    if (idioma=='ca') {
      document.writeln('Hola, ' + nomUsuari + '. ');
      if (comptador==1) document.writeln('És el primer accés que fas a
                                                    aquesta pàgina. ');
      else {
        document.writeln('És l\'accés número ' + comptador + ' que fas a
                                                    aquesta pàgina. ');
        document.writeln('L\'últim acces va esser el ' + dataUltimAcces + '.');
      }
    }
    if (idioma=='an') {
      document.writeln('Hello, ' + nomUsuari + '. ');
```

```
      if (comptador==1) document.writeln('This is the first access that you do to
                                                                   this web. ');
      else {
        document.writeln('This is the access number ' + comptador + ' that you do to
                                                                   this web. ');
        document.writeln('The last access was on ' + dataUltimAcces + '.');
      }
    }
    if (idioma=='fr') {
      document.writeln('Salut, ' + nomUsuari + '. ');
      if (comptador==1) document.writeln('C\'est la première visite que vous faites
                                                                   à ce site. ');
      else {
        document.writeln('C\'est la visite numéro ' + comptador + ' que vous faites
                                                                   à ce site. ');
        document.writeln('La dernière visite était le ' + dataUltimAcces + '.');
      }
    }
    crearCookie('nom', nomUsuari, '30');
    crearCookie('idioma', idioma, '30');
    crearCookie('comptador', ++comptador, '30');
    var ara=new Date();
    var dataHora = ara.getDate() + '/' + (ara.getMonth()+1) + '/' + ara.getFullYear()
          + ' -- ' + ((ara.getHours()<10)?('0'+ara.getHours()):(ara.getHours())) + ':' +
          ((ara.getMinutes()<10)?('0'+ara.getMinutes()):(ara.getMinutes())) + ':' +
          ((ara.getSeconds()<10)?('0'+ara.getSeconds()):(ara.getSeconds()));
    crearCookie('dataUltimAcces',dataHora, '30');
  }
</script>

<form name='form2'>
  <center>
    <input type='button' value='Esborrar totes les cookies' onClick=
                    'esborraCookie("nom"); esborraCookie("idioma");
                    esborraCookie("comptador"); esborraCookie("dataUltimAcces");'>
  </center>
</form>
 ...
```

Calaix desastre > **Passar paràmetres**

Un dels principals problemes en el disseny amb HTML és l'aïllament de les pàgines, que fa que no puguin passar-se informació. Bé, les pàgines sí que poden enviar dades, però no en poden recollir i processar. Amb JavaScript això té solució.

A la pràctica anterior hem vist com podem guardar dades a les Cookies i recuperar-les després. Aquesta és l'única opció que tenim, en JavaScript, per tenir dades persistents entre sessions. Això no funciona, és clar, si l'usuari ha desactivat les Cookies.

De vegades, però, el que necessitem és enviar unes dades des d'una pàgina i rebre-les a la següent, sense risc. En HTML tenim dos mètodes per enviar dades:

- **POST**: s'envien els camps d'un formulari, normalment, per correu electrònic.
- **GET**: s'envien les dades enganxades amb l'adreça, en format URL.

Aquest últim és fàcilment manipulable a partir de l'objecte **location**, que coneixem des de l'inici del curs. Es tractarà, doncs, d'analitzar l'adreça i extreure'n les dades.

Si, fins ara, havíem après com interactuar entre objectes d'una pàgina, entre finestres mares i filles, i entre marcs, amb aquesta pràctica tancarem el cercle: aprendrem a interactuar entre pàgines diferents, que poden estar situades, fins i tot, en servidors diferents.

Conceptes JavaScript en aquest capítol

Funció **escape() :** codifica una cadena en format hexadecimal.
Funció **unescape() :** descodifica una cadena hexadecimal codificada amb escape().
Mètode **substring() :** retorna una subcadena a partir d'un punt d'inici i una longitud.
Funció **isNaN() :** avalua una expressió i retorna "cert" si no és un número (is not a number).
Funció **parseFloat() :** analitza una cadena i, si es pot, retorna un número.
Mètode **replace() :** aplicat a una cadena, canvia uns caràcters per uns altres.
Expressió Regular : plantilla que s'utilitza per cercar patrons en cadenes.

1.- Passar un valor

En la declaració d'un formulari, si especifiquem el mètode GET, o no n'especifiquem cap, el botó "submit" passa l'adreça, el signe "?" i els parells "variable=valor". Sovint, però, l'únic que volem passar és un valor, que la pàgina receptora processarà i utilitzarà.

Tot i que podem utilitzar el mètode per defecte, tal com veurem al segon exemple, si les nostres necessitats són mínimes podem escriure un codi més senzill. En tot cas, el valor o els parells formaran part de l'adreça (location) que el navegador envia.

L'objecte location treballa amb URLs. Les adreces tenen aquesta estructura:

protocol :// host : port / pathname ? search # hash

Cadascuna d'aquestes és una propietat de location. Fixem-nos en **search**. S'utilitza per passar paràmetres als programes (CGI) o pàgines: asp, php,,, i JavaScript !!! Només cal posar un interrogant (?) a l'adreça. La cadena que segueix és search.

Per tal d'enviar aquesta cadena, el navegador "escapa" els caràcters no ASCII. La funció que els rebrà ho ha de preveure per retornar-los (**unescape**) al seu format original. També ha de tenir en compte que search comença al "?", i l'haurà de descartar. Per això, utilitzem el mètode **substring()**, que retorna una subcadena, començant per la posició especificada el primer paràmetre i que té la longitud del segon o, si no l'especifiquem, fins al final.

Necessitem, doncs, fer dues feines: a la pàgina origen, una funció que construeixi la cadena i la posi a la propietat **location.href**; a la pàgina destí, una línia que llegeixi l'adreça, descodifiqui la zona **search** i l'assigni a una variable.

Pàgina origen: passar

```
<script language="javascript">
function passar( url, param ) {
  location.href = url + '?' + escape( param )
}
</script>
</head>
<body>
<form name="form1">
<input type="text" name="txt" value="">
<input type="button" name="enviar" onClick="passar( 'pag2.htm', form1.txt.value )">
</form>
```

Pàgina destí (pag2.htm): llegir

```
<script language="javascript">
var param = unescape( location.search ).substring(1)
</script>
```

2.- Passar variables i els seus valors

A l'exemple anterior hem vist com passar un valor, però, i si el que volem és passar una llista de variables amb els seus valors? Doncs, en principi, la tècnica és la mateixa: enganxar a continuació del "?" i recollir-ho de la propietat **location.search**. Amb particularitats.

El primer que cal saber és com es codifica una cadena GET amb diversos paràmetres i, si ens hem fixat en les adreces dels cercadors, segur que ja ho sabem:

(adreça) **?** variable1=valor1 **&** variable2=valor2 **&** ...
(Hem posat espais en blanc per llegir-ho millor, però no hi han d'anar).

La funció que crearà l'adreça haurà de recollir - d'un formulari, un vincle o un altre lloc - uns valors per afegir a la cadena. Les variables de cada parell, en canvi, les escriurà com a literals. S'utilitzen com a referència "pactada", que el programa o pàgina receptors utilitzaran.

També hem de tenir en compte que el que passem és una URL. En conseqüència, hi haurà molts caràcters que no podrem utilitzar, com ara espais en blanc o accents. Afortunadament, tenim dues funcions que ho faciliten, **escape()** i **unescape()**, que s'encarreguen de codificar i descodificar una cadena a format hexadecimal, compatible amb URL.

El que farem és crear una cadena a partir de l'adreça de la pàgina. Primer li afegim '?'. Després, fem un vector a base de partir la cadena de noms amb el mètode **split()** i, amb un bucle, afegim a la cadena el nom de cada variable, el seu valor i el signe '&'. Finalment, esborrem l'últim signe de la cadena carreguem la pàgina amb la cadena.

Per tal d'assignar el valor de les variables hem utilitzat la funció **eval()**, que avalua una expressió, i, per assegurar que no hi ha caràcters estranys, hem passat el resultat per **escape()**. A partir d'ara, podem utilitzar la funció des de JavaScript o HTML.

Pàgina origen: passar

```
<script language="javascript">
function pasParams( pag, noms ) {
  pag += '?'
  vect = noms.split( ',' )
  for ( i=0; i<vect.length; i++ ) {
    pag += vect[i] + '=' + escape( eval( vect[i] ) ) + '&'
  }
  pag = pag.substring( 0, pag.length-1 )
  location.href = pag
}
var v1 = "Això és un text"
var v2 = 166.386
</script>
</head>

<body>
<a href="javascript:pasParams( 'pag2.htm', 'v1, v2' )"><b>Enviar</b></a>
```

...

A la pàgina destí també hi haurà una mica de codi, que haurem de posar a la capçalera, de manera que puguem processar i utilitzar les dades rebudes des de l'inici.

En primer lloc, eliminem el signe d'interrogació de la cadena search, com a l'exemple anterior. Després, fem un Array a base de partir els paràmetres pel signe '&'. A continuació, fem un bucle sobre aquest Array, partim cada element pel signe '=' i assignem a cada variable el seu valor:

Per tal d'assegurar que cada variable rebrà valors del tipus que espera, hem fet un condicional: si el valor no és numèric - isNaN(parseFloat(aVar[1])) - hem d'afegir-hi cometes, si ho és, no cal. És un sistema arriscat: si volem passar paràmetres literals que contenen números (p.ex. de telèfon o DNI), s'interpretarien com a numèrics. En aquests casos ho hauríem de preveure i escriure les línies de correcció de casos concrets.

Pàgina destí (pag2.htm): llegir

```
<script language="javascript">
var param = location.search.substring(1)
aParams = param.split( '&' )
for ( i=0; i<aParams.length; i++ ) {
  aVar = aParams[i].split("=")
  if ( isNaN( parseFloat( aVar[1] ) ) ) eval( aVar[0] + '="' + unescape( aVar[1] ) + '";' )
  else eval( aVar[0] + '=' + aVar[1] + ';' )
}
</script>
```

3.- Enviar un formulari a la pròpia pàgina

Es tracta d'una variació de l'exemple anterior. En aquest cas, les pàgines origen i destí són la mateixa. Aquesta és una tècnica utilitzada moltes vegades, i en podem veure un exemple al **cercador intern** d'aquest curs.

L'únic que ens cal és tenir una mica d'ordre a l'hora de dissenyar els scripts:

1.- Si cal, declarem les variables a l'inici. Depèn de si passem només un valor o parells.
2.- Comprovem si, amb l'adreça, es reben paràmetres. Si és així, els assignem.
3.- Com que sabem quants i quins paràmetres es reben, l'assignació serà més senzilla.

En aquest exemple utilitzem el mètode GET, establim com a acció (action) la pròpia pàgina i posem un botó "submit" que l'envia. Així haurem vist tres models. Val a dir que, tots tres ("button" més funció, vincle més funció i GET) són vàlids, i l'elecció és qüestió de gustos.

L'avantatge d'aquest mètode és, per una banda, que podem usar "submit" en lloc de "button" (no cal definir l'event onClick) i, per l'altra, que ens estalviem una funció.

El mètode GET envia parelles camp=valor. Les variables, doncs, tenen el nom dels camps del formulari. Els valors s'envien en un format diferent: en lloc d'espais s'utilitza el signe '+'. Això va molt bé als CGIs, però, en el nostre cas, és un problema. El solucionem utilitzant el mètode de l'objecte String **replace()**, que permet canviar uns caràcters per uns altres. El primer paràmetre ha de ser una expressió regular. Què?

Una **expressió regular** és una mena de plantilla que s'utilitza per cercar patrons en cadenes. Té una zona per definir el patró i una pels modificadors: **/ patró / modificadors**. En el nostre cas volem que es substitueixin tots els signes "+" per espais blancs. el patró serà **\+** (usem l'antibarra per indicar que és un caràcter, no un operador) i el modificador, "global":

/ \ + / g vol dir: *el caràcter "+", globalment*.

> L'altre modificador habitual és **/i**, que vol dir que s'ignorin majúscules i minúscules. Es poden escriure tots dos junts, **/gi**. En aquest exemple, però, no cal. Conèixer i aprendre totes les possibilitats de les expressions regulars és una tasca llarga i feixuga i, en tot cas, no forma part dels objectius d'aquest curs.

Finalment, el codi que controla la recepció i assignació de paràmetres es basa en l'exemple anterior, però és més senzill - i menys universal - perquè sabem el què s'espera.

```
<html>
<head>
<title>Exemple</title>
<script language="javascript">
var txt1 = '', txt2 = ''
var param = location.search.substring(1)
if (param>''){
  aParams = param.split( '&' )
  for ( i=0; i<2; i++ ) {
    aVar = aParams[i].split("=")
    eval( aVar[0] + '="' + unescape( aVar[1].replace( /\+/g, ' ' ) ) + '";' )
  }
}
</script>
</head>

<body bgcolor="#EEEECC">
<h3>Enviar paràmetres a la pròpia pàgina</h3>
<form name="form1" method="GET" action="m8p2_3.htm">
Camp 1: <input type="text" name="txt1" value=""><br>
Camp 2: <input type="text" name="txt2" value="">      
<input type="submit" value="Enviar">
</form><hr>
<h3>Els valors rebuts són:</h3>
<script language="javascript">
document.writeln( 'Camp 1: ' + txt1 + '<br>Camp 2: ' + txt2 )
</script>
</html>
```

La creació d'aplicacions complexes amb JavaScript comporta l'ús d'una metodologia específica. Si intentem abastar un problema complex sense seguir uns mètodes apropiats ens podem trobar desbordats i sense poder resoldre el problema.

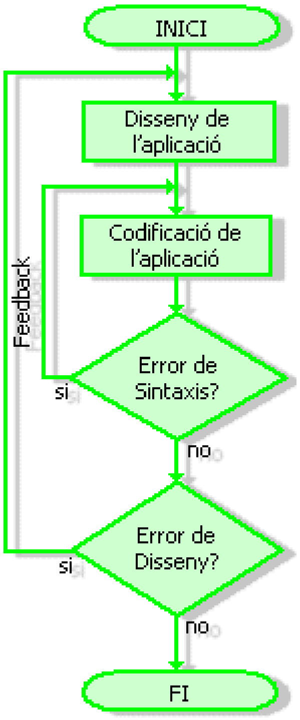
La creació d'una aplicació complexa consta de dues fases principals: el disseny i la depuració.

Abans de començar a escriure un programa extens és necessari dedicar un temps a pensar com s'ha de resoldre el problema. L'enfocament més apropiat és la **programació estructurada**. Entendre els principis d'aquest tipus de programació és senzill: Divideix i venceràs. És a dir, fem el problema global i complex en petits trossos més assequibles.

Dins de la programació estructurada hi ha diferents formes d'actuar. El **disseny descendent** és la més utilitzada.

Un cop s'ha fet el disseny s'ha de passar a la implementació. Per això, haurem de codificar els blocs dissenyats i executar-los. En aquesta fase ens trobarem que sorgiran múltiples errors i per solucionar-los s'hauran d'aplicar els **mètodes de depuració**.

La detecció d'errors i la seva resolució ens obligaran a efectuar un feedback entre les fases de disseny i depuració. El codi l'haurem de repassar múltiples vegades fins a obtenir-ne el resultat desitjat.



En aquesta pràctica veurem els principals conceptes aplicables a les fases de disseny i depuració d'errors. Per si mateixos no seran suficients per poder-los aplicar a qualsevol tipus de problema, ja que és necessari disposar d' una certa habilitat i experiència en la programació, i això, solament s'aconsegueix amb el temps.

Conceptes de programació en aquest capítol

Concepte de **Programació estructurada**.
Concepte de **Disseny descendent**.
Tècniques de **depuració** del codi.

1.- La fase de disseny. Concepte de disseny descendent

La metodologia de disseny més acceptada dins la Programació Estructurada és la del **disseny descendent**. A partir d'un problema complex, es desenvolupa en blocs operatius enniuats que cada cop seran més concrets. Cadascun d'aquests blocs serà una funció que rebrà paràmetres i retornarà valors.

A la vegada cada bloc es subdivideix en altres blocs que també seran funcions que rebran paràmetres i retornaran valors, i així successivament.

D'aquesta forma s'aconsegueix passar d'un bloc inicial més abstracte a uns altres blocs en nivells inferiors enniuats, més concrets.

Vegem-ho millor amb un exemple:

Suposem que desitgem fer un programa que serveixi per calcular la nota mitjana d'un grup d'alumnes.

De cada alumne tenim les seves notes parcials i la ponderació dins un array amb la

següent estructura:

```
var dades = new Array (
Grup; nNotes; pondNota1; pondNota2; pondNota3; .....
nomAlumne1; nota1; nota2; nota3; ...
nomAlumne2; nota1; nota2; nota3; ...
.....
);
```

Desitgem extreure la informació d'aquest array de forma que s'obtingui finalment la nota mitjana global de tot el grup.

A grans trets, el nivell d'anàlisi més elevat i més abstracte constaria dels següents passos:

1. Llegir el nom del grup i posar-lo en una variable.
2. Llegir el número de notes i posar-les en una variable.
3. Llegir la ponderació de cada nota i posar-la en una variable-array.
4. Obtenir la nota mitjana de tot el grup.
5. Mostrar la nota mitjana de tot el grup.

El punt 4 representa un concepte molt extens, la seva implementació s'ha de desglossar en apartats més senzills. Seria convenient aïllar-lo i resoldre'l a part mitjançant una funció especialitzada en aquesta tasca.

En un primer nivell de concreció el punt 4 el podem dividir en:

4.1. Obtenir la nota mitjana d'un alumne.
4.2. Acumular la nota mitjana de tots els alumnes.
4.3. Repetir els passos 4.1 i 4.2 fins a l'últim alumne.
4.4. Dividir la suma acumulada de notes mitjanes pel nombre d'alumnes.
4.5. Retornar la nota mitjana de tots els alumnes.

Tots els apartats 4.x indicats prèviament seran una funció que rebrà com a paràmetres el número de notes per alumne i la ponderació de cada nota. Un cop efectuats els passos corresponents, retornarà la nota mitjana de tots els alumnes.

L'apartat 4.1, també és complex i per tant, també el farem més concret mitjançant una altra funció més específica, amb les mateixes condicions que l'anterior.

En un segon nivell de concreció el punt 4.1 anterior es pot dividir en:

4.1.1. Obtenir el nom de l'alumne
4.1.2. Obtenir una nota de l'alumne i acumular-la a una variable.
4.1.3. Repetir el pas anterior fins a l'última nota.
4.1.4. Retornar el valor de la nota ponderada mitjana de l'alumne.

Finalment, el pas 4.1.2 anterior també el podem especificar mitjançant una altra funció en un nivell de concreció més específic. Per tant, en un tercer nivell de concreció, el pas 4.1.2 es pot dividir en:

4.1.2.1. Obtenir una nota de l'alumne i multiplicar-la per la ponderació.
4.1.2.2. Retornar la nota de l'alumne amb la ponderació.

Observem que s'ha anat del més abstracte al més concret. És a dir, hem desglossat el problema general en petits problemes, cada cop més assequibles. Cada nivell de concreció que s'aprofundeix està format per una o més funcions i cada funció rep uns paràmetres i retorna un resultat.

Tot i que no se n'ha parlat fins ara, aquest mètode és el que s'ha utilitzat en tot el curs. Però, per a problemes molt més complexos, aquest sistema es torna ineficaç. És en aquesta situació on es fa necessari aplicar els conceptes de la POO (Programació Orientada a Objectes), que ja no s'explicarà aquí per la seva gran extensió.

Les versions actuals de JavaScript no implementen la POO completament (tot i que constantment es treballa amb mètodes i propietats d'objectes) encara que mitjançant la propietat *prototype* és possible crear models equivalents a les classes de la POO.

2.- La fase de Depuració del codi

Un cop efectuada la fase de disseny indicada a l'apartat anterior, és necessari convertir cada nivell de concreció en les funcions corresponents mitjançant el codi JavaScript i provar el seu funcionament. És a dir, hem de **codificar** el disseny efectuat.

És en aquest moment que ens apareixen els **errors**, generalment inevitables quan el codi és mitjanament extens. Ara és quan hem de fer la **depuració del codi**. En aquesta fase tenim dos tipus d'errors: els *errors de sintaxi* i els *errors de disseny*.

Errors de sintaxi.
Els errors de sintaxi són els originats per la introducció incorrecta de les ordres. Aquests tipus d'errors ens els detecta el mateix navegador i són deguts, generalment, a errades de tecleig a l'hora d'escriure els comandaments o a errors per paritats incorrectes en escriure { } , ( ) o " ", etc. La forma d'indicar l'existència d'un d'aquests errors és diferent per IE o NN.

IE ens mostra el símbol

a l'esquerra de la barra d'estat. En fer clic a sobre del símbol apareix una finestra amb el número de línia i el tipus d'error.

NN ens mostra el text

a la barra d'estat del navegador. Llavors, hem d'escriure el text '**javascript:**' a la barra d'adreces del navegador i ens apareixerà una finestra amb la indicació del número de línia que ha provocat l'error i el tipus d'error.

Un cop sabem el tipus d'error i número de línia hem de mirar de trobar l'error a la línia indicada i solucionar-lo. Aquest pas pot ser complex, ja que a vegades l'error no es troba en la línia indicada sinó en una línia anterior (per exemple, obrir unes cometes i no tancar-les, o un parèntesi, etc.). També es pot donar el cas que s'hagin produït múltiples errors. En aquesta situació, és aconsellable solucionar el primer error i tornar a executar. Moltes vegades els següents errors són conseqüència del primer de tots.

Errors de disseny.
Un cop hem solucionat tots els error de sintaxi, hem de comprovar si el nostre programa fa el que n'esperem. Si no és així, vol dir que tenim errors de disseny. Aquests tipus d'errors són difícils de localitzar i, generalment, es fa necessari efectuar una anàlisi del codi pas a pas. Els llenguatges de programació convencionals disposen dels programes debugger o depuradors que permeten efectuar una anàlisi detallada del funcionament del programa.

La forma de depurar el codi per trobar els errors de disseny pot ser mitjançant "xivatos" o mitjançant el depurador.

Els "xivatos" consisteixen en trossos de codi afegits dins el codi a depurar per mostrar, estàticament (aturant l'execució del codi) o dinàmicament (sense aturar l'execució del codi) el valor de variables en punts determinats de la seqüència d'execució.

El "xivato" estàtic utilitzat habitualment és el mètode *alert()* i el dinàmic més habitual és la barra d'estat (*window.status*).

El següent exemple de codi correspon a la funció **eliminaBlancsExtrems()** utilitzada al mòdul 7, pràctica 2, apartat 1. S'hi ha introduït un error de disseny intencionadament (s'ha substituït la condició **cadena.charAt(*0*)==' '** per **cadena.charAt(*1*)==' '** dins el primer *while*). Aquest error provoca que no s'eliminin tots els espais en blanc inicials quedant-ne sempre un.

Mitjançant tres xivatos temporals implementats amb *alert()* podem fer el seguiment pas a pas de l'evolució de la variable *cadena* i determinar en quin pas funciona malament. Un cop detectat l'error de disseny i solucionat, els xivatos s'hauran d'eliminar.

En executar el codi següent, observarem com el "Xivato 1" s'activa una vegada menys que el nombre d'espais en blanc introduïts al principi. Això ens ha de fer veure que dins aquest *while* hi ha un error que provoca que sempre quedi un espai sense eliminar al principi.

```
function eliminaBlancsExtrems(cadena)
{
  while (cadena.charAt(1)==' '){
    alert('Xivato 1: Eliminació blancs inicials de la cadena:\n\'' + cadena + '\'');
    cadena = cadena.substring(1,cadena.length);
  }
  while (cadena.charAt(cadena.length-1)==' ') {
    alert('Xivato 2: Eliminació blancs finals de la cadena:\n\'' + cadena + '\'');
```

```
    cadena = cadena.substring(0,cadena.length-1);
  }
  alert('Xivato 3: Resultat final:\n\"' + cadena + '\"');
  return cadena;
}
```

La utilització de la barra d'estat ens pot servir per implementar un "xivato" que ens mostri l'estat d'algunes variables d'un programa o bé el pas per determinats llocs del programa de forma dinàmica, és a dir, sense haver de parar l'execució del programa. El següent exemple és el mateix que l'anterior, amb el mateix error.

En aquest cas, no s'atura l'execució del codi, però, a la barra d'estat del navegador ens indica cada cop que passa per l'interior d'un dels blocs *while* i al final veiem com ha quedat la cadena introduïda. Per cada espai eliminat al principi apareix un símbol ' ***** ' i per cada espai eliminat al final apareix un símbol ' **+** '.

Podem observar com el nombre de vegades que passa pel primer *while* és sempre una menys que el nombre d'espais introduïts a l'inici de la cadena. Per tant, es dedueix que aquesta part de codi no funciona correctament com a conseqüència de l'error de disseny que hi hem introduït intencionadament.

A l'igual que l'exemple anterior, un cop detectat i solucionat l'error, s'han d'eliminar els "xivatos" introduïts.

```
function eliminaBlancsExtrems(cadena)
{
  while (cadena.charAt(1)==' '){
    window.status += " * ";
    cadena = cadena.substring(1,cadena.length);
  }
  while (cadena.charAt(cadena.length-1)==' ') {
    window.status += " + ";
    cadena = cadena.substring(0,cadena.length-1);
  }
  window.status += " Resultat: '" + cadena +"'";
  return cadena;
}
```

3.- Altres opcions de depuració.

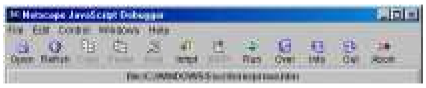
En aquest apartat comentarem lleugerament dues utilitats interessants per efectuar la depuració de codi: el *provador de codi* incorporat a les eines d'aquest curs i un *depurador* fet en Java per Netscape.

El **provador de codi** que tenim en aquests curs és molt útil per provar petits blocs de codi abans d'incorporar-los dins el programa. Per utilitzar-lo, es tracta d'introduir el bloc de codi en el quadre de text, mitjançant l'escriptura directa o bé amb *copiar* i *enganxar* i prémer sobre el botó "Provar Script". Posteriorment veurem una finestra amb el resultat de l'execució del codi. Podem efectuar modificacions en el codi fins a obtenir el resultat desitjat.

Una altra opció per efectuar la depuració és la utilització d'un **programa depurador**, a l'estil del que s'utilitza en altres llenguatges tipus C/C++, Pascal, Visual Basic, etc. Aquest depurador l'ofereix Netscape i solament funciona en aquest navegador. A més, és imprescindible tenir disponible el Java per al navegador (compte, el NN6.x no el porta incorporat per defecte i s'ha de descarregar de la web per instal·lar-lo).

Aquest depurador el podeu trobar a l'adreça http://developer.netscape.com/software/jsdebug.html, disponible per a vàries versions del navegador NN. Per si no hi fos, aquí en teniu una de les primeres versions, la versió JavaScript Debugger 1.1 per a NN 4.0 o superior.

Per utilitzar-lo l'hem de descomprimir dins una carpeta i amb el NN obrir el fitxer "*jsdebugger.html*" . Automàticament s'executa un programa Java que ofereix les opcions de depuració.



A la finestra del navegador NN s'ha d'obrir la pàgina

HTML que conté el codi JavaScript que volem depurar. Aquesta finestra serà la que utilitzarem per executar el codi. Posteriorment, dins la finestra del debugger hem d'obrir la mateixa pàgina HTML. Automàticament quedaran identificades les parts de codi JavaScript diferenciant-lo del codi HTML mitjançant una barra lateral de color groc.

En el tros de codi senyalitzat amb la barra lateral podrem indicar el punts de parada de l'execució (breakpoints). Quan executem el codi dins la finestra del navegador NN corresponent als llocs on hi hagi un breakpoint, aquest s'aturarà i ens permetrà efectuar la visualització del valor de les variables, executar pas a pas, etc.

S'ha de tenir en compte que el depurador que tenim disponible aquí reconeix les instruccions del NN4. Per a d'altres navegadors superiors de Netscape és necessari descarregar el depurador corresponent si desitgem que es reconeguin les noves prestacions aportades. Si el que volem depurar és codi que no implementi objectes propis d'un determinat navegador, aquest depurador ens serà perfectament vàlid.

Les dues eines indicades permeten més opcions que les que s'han comentat. Però, aquí no és possible descriure-les completament. Amb la pràctica es van descobrint totes les seves possibilitats.

Bé, aquesta és l'última pràctica del curs i segur que ens hem deixat molts aspectes sobre DHTML i JavaScript sense comentar o aprofundir. Entendre i aprendre un llenguatge no és el mateix que dominar totes les seves possibilitats. Això requereix molt de temps, potser anys, fins i tot hi ha qui pensa que no s'acaba mai.

Sigui com sigui, tenim la base suficient per donar un gir espectacular a les nostres pàgines i continuar aprenent. Ara, el que toca és ser pràctics i veure com podem rendibilitzar al màxim el que sabem fer i el que anirem incorporant. Tots els exemples d'aquesta pràctica són candidats a formar part de moltes pàgines.

Conceptes JavaScript en aquest capítol

Propietat **prototype :** permet afegir, esborrar o modificar propietats i mètodes d'un objecte.
Comportament **onerror :** determina què s'ha de fer si es produeixen errades.
Mètode **sidebar.addPanel() :** afegir una adreça als preferits (Netscape 6+, Mozilla)
Mètode **external.AddFavorite() :** el mateix, però amb Internet Explorer.

1.- Enllaçar arxius externs

Si volem utilitzar bocins de codi en diversos llocs, el més pràctic és que els guardem en arxius i els enllacem a les pàgines. Això, que fem per força amb les imatges, els sons, els vídeos o el Flash - tots són bits - ho podem fer també amb arxius de text.

Un script contingut en un arxiu s'enllaça utilitzant l'etiqueta <script> amb el modificador **src** assignat a l'arxiu. També hi podem afegir el tipus o el llenguatge utilitzat:

```
<script language="JavaScript" type="text/javascript" src="arxiu.js"></script>
```

Si l'arxiu enllaçat només conté funcions - sense codi que s'hagi d'executar mentre es llegeix -, podem forçar que es vagi carregant "de fons", mentre es va llegint la pàgina, utilitzant el modificador **defer**. Així accelerem la càrrega de la pàgina.

```
<script defer language="JavaScript" type="text/javascript" src="arxiu.js"></script>
```

Per enllaçar un arxiu de definició d'estils utilitzem l'etiqueta <link> amb l'arxiu assignat a **href**. A més, definim la relació, el tipus i el medi:

```
<link rel="stylesheet" href="pantalla.css" media="screen" type="text/css">
```

L'etiqueta <link> pot tenir altres utilitats i altres modificadors. Els navegadors no en fan ús, però no es descarta que els utilitzin en futures versions. Alguns cercadors, en canvi, sí que els poden utilitzar per crear una mena de "mapa del lloc". Alguns d'aquests modificadors són: target (finestra destí), hreflang (idioma), charset (conjunt de caràcters), ...

El modificador "rel" pot tenir, al seu torn, diversos valors: start, prev, next, help, glossary, contents, copyright, fontdef, alternate, ... De tots els possibles, els navegadors utilitzen l'últim, **alternate**, per especificar un full d'estils alternatiu. Amb això i alguns scripts podríem, per exemple, fer pàgines personalitzades. Aquest és un tema "d'última fornada", massa extens per ser tractat aquí. Això sí, sabem que hi és i, a partir d'ara, el trobarem cada cop en més pàgines.

Com a curiositat, i també com a component útil, tenim un arxiu, **styleswitcher.js**, que proporciona tota la funcionalitat per utilitzar fulls d'estils alternatius. El seu autor és un jove britànic, Paul Sowden. Més informació a la seva web: **http://idontsmoke.co.uk**

2.- Llibreries

Alguns llenguatges de programació avançats utilitzen llibreries. Es tracta de col·leccions de funcions organitzades que s'enllacen de forma dinàmica - només les que es necessiten - amb el codi que escrivim, en el moment de la compilació, per generar els executables.

JavaScript no es compila i no disposa de llibreries. Però podem crear, organitzar i guardar arxius amb les funcions més comunes, de manera que, quan les necessitem, només ens calgui enllaçar-les amb el font, o "copiar i enganxar" a les nostres pàgines.

De fet, la creació de llibreries no està limitada al JavaScript. Podem fer-ne també de fulls d'estils, d'estructures HTML o, com vam fer al primer mòdul, d'estructures de carpetes. Tot allò que puguem tornar a necessitar, ho podem guardar i considerar una llibreria.

Com que aquestes col·leccions, amb el temps, poden anar creixent, és fonamental tenir una bona organització i molta claredat. Això sí, si ens dediquem a inserir comentaris al propi codi, ocuparem un espai i un temps de càrrega innecessaris. És preferible acompanyar cada carpeta amb un arxiu de text que inclogui explicacions sobre el seu contingut.

Pel que fa als scripts, els podem escriure com vulguem, provar, rectificar, ... Un cop acabats - hem d'estar segurs que ja no els modificarem més -, hauríem de fer que ocupin el mínim espai. Per això, guardem la funció en el seu format original (amb comentaris, si cal) i en fem una còpia. Podem anomenar els originals, per exemple, "xxxx.ori" i les còpies "xxxx.js".

A la còpia, substituïm els noms de les variables, els deixem només amb una lletra, eliminem els espais innecessaris i canviem els salts de línia pel signe ";" (punt i coma). Queda com una botifarra de signes, difícil de llegir. És el més semblant a una compilació.

Com a exemple, fem una funció per formatar els milers. Rebrà un número (amb o sense decimals), el convertirà en cadena, separarà enters i decimals, partirà els enters en grups de tres - començant per la cua - i ens tornarà la cadena amb els milers separats per punts i els decimals per una coma.

Primer hem escrit la funció amb variables normals. Un cop provada, les hem canviades per variables d'una lletra (ho podem veure al codi) i, finalment, hem tret els espais. La funció ha passat dels 523 bytes inicials als 232 finals, és a dir un 45% del que ocupava. A més, el codi resultant és més difícil d'interpretar i, en conseqüència, té menys interès per a la pirateria.

Com hem escrit la funció (amb variables d'una lletra):

```
<script language="javascript">
function formatMil(x) {
  x += ''
  var a = x.split( '.' )
  var n = a[0]
  var l = n.length
  var c= ''
  if (parseInt(n) + '' == n) {
    while (l>3) {
      c = "." + n.substr( l-3, 3 ) + c
      n = n.substr( 0, l-3 )
      l = n.length
    }
    if (n.length>0) c = n + c
    if (x.indexOf( '.' )>0) c += ',' + a[1]
  }
  return c
}
</script>
```

Funció preparada per formar part d'una llibreria:

```
function formatMil(x){x+='';var a=x.split('.');var n=a[0];var l=n.length;var c='';
if(parseInt(n)+''==n){while(l>3){c="."+n.substr(l-3,3)+c;n=n.substr(0,l-3);l=n.length}
if(n.length>0)c=n+c;if(x.indexOf('.')>0)c+=','+a[1]}return c}
```

3.- Noves funcionalitats per a vells objectes

Els llenguatges orientats a objectes - Java, C++ - basen la seva herència en classes. Els objectes JavaScript, en canvi, utilitzen l'herència basada en prototips. Els constructors creen instàncies basades en les propietats i mètodes definits al prototip dels objectes.

Tots els objectes tenen la propietat **prototype**, que permet afegir, esborrar o modificar propietats i mètodes de l'objecte original, de manera que les noves instàncies els heretin. Això ens permet, per exemple, atorgar als objectes noves funcionalitats.

Afegirem quatre mètodes nous a les cadenes (objecte String). Són típics d'altres llenguatges, de manera que molts programadors hi estan acostumats. Els dos primers, **left(n)** i **right(n)**, retornen un bocí de la cadena, per l'esquerra o per la dreta, amb la longitud que indiquem. Utilitzen el mètode **substr()** (també podem escriure "substring"), que ja coneixem.

El següent mètode, **trim()**, elimina els espais en blanc que hi pugui haver a l'inici i/o la fi d'una cadena, utilitzant una expressió regular. L'últim mètode, **rep(n)**, retorna una cadena a base de repetir els caràcters de què està formada, el número de vegades que especifiquem.

De passada, veurem una nova manera de declarar mètodes i funcions, recomanada pel W3C, l'organisme que regula els estàndards d'Internet:

Objecte**.prototype.**metode = **function(** paràmetres **) {** … *instruccions* … **}**

Aquests quatre mètodes funcionen. Recordem, però, que si els volem incorporar en forma de llibreria, hauríem d'eliminar abans els espais que sobren.

```
<script language="javascript">
String.prototype.left = function(n) {
  return this.substr( 0, n )
}
String.prototype.right = function(n) {
  return this.substr( this.length - n, n )
}
String.prototype.trim = function() {
  return this.replace( /^\s*(\b.*\b|)\s*$/, '$1')
}
String.prototype.rep = function(n) {
  var s = '', t = this.toString()
  while (--n >= 0) s += t
  return s
}
</script>
```

4.- Control d'errades

Amb les errades passa una cosa curiosa: sempre n'hi ha. Quan es mostra el missatge que proporciona el navegador, fa mal efecte. En canvi, si el missatge és personalitzat, la nostra reputació com a programadors millora. Doncs, bé; com que no sempre podem evitar-les, almenys, aprofitem-les.

L'exemple següent mostra un mecanisme per informar sobre les errades, molt simple. L'ideal seria incloure-hi un formulari i un botó, amb el prec que es premi per informar-nos sobre les incidències. Això sí que és donar-li la volta a la truita!

Per començar, desviem les errades cap a una funció: window.**onerror** = eAfegir. A continuació declarem tres Arrays que contindran, respectivament, el missatge, l'adreça i la línia de l'errada. El navegador passa aquests paràmetres a la funció de forma automàtica.

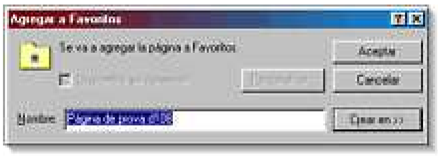
La funció receptora, senzillament, afegirà elements a cada Array amb els valors rebuts. Per això s'utilitza la propietat length, que és l'índex de l'últim element. La funció retorna "true" i, com ja sabem, s'evita que s'engegui el comportament per defecte.

Farem encara una altra funció que mostri l'informe. Crearem una finestra on escriurem, amb l'ajut d'un bucle, els valors dels tres Arrays. Finalment, tanquem el document de la finestra. Això és el mínim. Com ja hem dit, aquí s'hauria de donar bona impressió i demanar a l'usuari que ens informi sobre les errades. Podem aprofitar per incloure-hi altres informacions, que ja hem vist als **exemples** del mòdul anterior (mòdul 7, pràctica 3, exemples 1 i 2).

```
<script language="javascript">
window.onerror = eAfegir
aMsg = new Array()
aUrl = new Array()
aLin = new Array()
function eAfegir(msg, url, lno) {
  aMsg[aMsg.length] = msg
  aUrl[aUrl.length] = url
  aLin[aLin.length] = lno
  return true
}
function eMostrar() {
  winError=window.open('','error','scrollbars=yes')
  with ( winError.document ) {
    writeln('<b>Informe d\'errades:</b><br><br>')
    for (var i=0; i < aMsg.length; i++) {
      writeln('<b>Arxiu:</b> ' + aUrl[i] + '<br>')
      writeln('<b>Línia:</b> ' + aLin[i] + '<br>')
      writeln('<b>Error:</b> ' + aMsg[i] + '<br><br>')
    }
    close()
  }
}
</script>
</head>

<body ...>
<a href="javascript:eMostrar()">Informe d'errades</a>
...
```

5.- Afegir la pàgina als marcadors preferits

De vegades cal fer una feina que afecta propietats específiques del navegador i que tenen poc a veure amb la pàgina. És el cas de la funció que ara presentem. Com que cada navegador utilitza funcions diferents (fins i tot, nom diferents) per als "Favoritos", haurem de comprovar primer l'existència d'uns objectes determinats i, després, escriure les funcions adients.



A partir de la versió 6 de Netscape s'utilitza l'objecte **window.sidebar** i el mètode **addPanel()**. Internet Explorer fa servir el mètode **external.AddFavorite()**. Finalment, les versions de Netscape anteriors a la 6 no ofereixen cap possibilitat, l'únic que podem fer és presentar un missatge.

```
<script language="javascript">
function afegirFav() {
  var url = location.href
  var nom = document.title
  if ( window.sidebar && window.sidebar.addPanel ) {
    window.sidebar.addPanel( nom, url, '' )
  }
  else {
    if ( document.all ) window.external.AddFavorite( url, nom)
    else alert( 'Premeu Control + D per afegir la pàgina als marcadors' )
  }
}
</script>
</head>

<body ...>
<a href="javascript:afegirFav()">Afegir als marcadors</a>
...
```

Ja som a l'últim exercici del curs, que tracta dels continguts de les pràctiques d'aquest mòdul, però, heu de procurar que siguin adaptables per al vostre projecte. Si ho considereu convenient, podeu afegir altres funcionalitats que pugueu necessitar, mantenint sempre els mínims que es demanen aquí.

Abans de començar modificarem la pàgina *index.htm* que teniu dins la carpeta *m8* i hi introduireu un enllaç, que ens porti a la primera pàgina de l'exercici.

Exercici

Consisteix en la realització de dues pàgines HTML, la segona serà enllaçada des de la primera.

Fer una pàgina que recordi el nom de l'usuari i la data de l'últim accés. En el primer accés a la pàgina, si les cookies no existeixen, s'ha de sol·licitar el nom a l'usuari. Posteriorment, s'han de desar les cookies corresponents amb un temps d'expiració elevat.

En els propers accessos a la pàgina, hem de mostrar una salutació personalitzada i la data de l'últim accés a la capçalera de la pàgina. La resta de la pàgina pot contenir informació pròpia del vostre projecte.

Com a element auxiliar, per fer proves, afegirem un botó que permeti esborrar les cookies creades per aquest document (aquest botó s'haurà d'eliminar quan incorporeu el contingut d'aquest exercici al vostre projecte).

La pàgina ha de disposar d'un altre botó. En fer-li clic, ha d'obrir una segona pàgina que rebi com a paràmetres mitjançant GET el nom d'usuari, la data d'últim accés i la url des d'on s'ha cridat (location.href) i també els mostri a la capçalera de la pàgina.

L'objectiu consisteix que un conjunt de pàgines (en aquest exercici, solament dues) d'un mateix lloc web, mostrin constantment el nostre nom, el darrer accés i d'on venim. D'aquesta forma es produeix l'efecte com si les pàgines haguessin estat creades personalitzades per a nosaltres.

Si al cap d'un temps es torna a visitar el mateix lloc web, pel fet de mostrar el nostre nom directament amb informació personalitzada, serà més impactant per a l'usuari.

Les pàgines han d'estar formatades mitjançant els mateixos estils CSS. Tots els estils CSS i el codi JavaScript no han de formar part de les pàgines directament, sinó que han d'estar inclosos i s'han d'enllaçar adequadament, mitjançant fitxers externs amb extensió .css i .js .

Adapteu aquestes pàgines d'exercici i el full d'estils perquè puguin ser vàlides per al vostre projecte.

Trucs, consells i receptes

Aprofiteu les funcions disponibles a la pràctica 1 per treballar amb cookies. Introduïu aquestes funcions directament al fitxer .js. Us podran servir per a altres aplicacions com si es tractés de funcions de llibreria.

Envieu un missatge a la vostra tutoria un cop l'exercici estigui penjat a Internet. Indiqueu sempre l'adreça de forma que s'hi pugui accedir directament.

Recomanacions per al projecte

El projecte ha de servir per sintetitzar els nous coneixements adquirits durant el curs i a la vegada, utilitzar-los en la realització d'una aplicació web que pugui ser útil per a la nostra tasca docent o personal.

La correcta realització del projecte, conjuntament amb els exercicis dels mòduls, ens permetrà finalitzar satisfactòriament el curs i disposar de la certificació corresponent.

El projecte ha de ser personal i inèdit.

El treball sobre el projecte comença amb la proposta del tema al tutor al final del mòdul 5. La realització d'aquesta proposta ha de seguir les indicacions que es donen en aquesta **pàgina**. A partir del mòdul 7, els exercicis que es realitzaran s'han d'orientar per formar part del projecte.

A grans trets, el projecte ha de reunir les següents **característiques**.

- La **interactivitat** entre l'usuari i la web. Hem vist al llarg del curs múltiples exemples on l'usuari "controla" el que desitja veure, el nostre projecte n'ha de ser un bon exemple.

- La **presentació** ha de ser entenedora i agradable per l'usuari i s'ha de deduir ràpidament la seva utilitat. D'aquesta manera evitarem que sigui rebutjada per l'usuari en no poder-la entendre.

- La **uniformitat** en els estils s'ha d'aconseguir mitjançant un full d'estils comú a totes les pàgines.

- La **reutilització** també és imprescindible. Per exemple, si fem una aplicació per mostrar les fotos del viatge de final de curs, el curs següent també hem de poder utilitzar la mateixa aplicació canviant únicament les imatges, descripcions i estils que podran estar definits en un únic arxiu de text. Hem de fer que tot el codi i estructura de les pàgines es puguin tornar a aprofitar gairebé sense modificar-les.

- La **globalitat** dels continguts. S'ha de procurar utilitzar la majoria dels continguts estudiats durant el curs. Com a mínim hem d'implementar algun dels conceptes corresponents a cadascun dels blocs temàtics del curs. A més, han d'estar contextualitzats amb la resta d'elements de la web.

En la **valoració del projecte** es tindrà en compte que incorpori el màxim de solucions possibles mitjançant JavaScript i DHTML. Sempre que sigui possible i contextualment justificable, s'han d'aplicar els conceptes descrits en aquest curs. Si algun apartat es pot fer amb HTML o amb JavaScript, seleccionarem preferentment l'opció de JavaScript.

L'aplicació web ha d'ésser **compatible** amb els principals navegadors (Netscape Navigator, Mozilla, Internet Explorer). Si alguna part no és possible compatibilitzar-la, ho hem de detectar, i si s'escau, mostrar alguna altra opció alternativa que tingui la mateixa funcionalitat. Hem d'evitar que apareguin errors per incompatibilitat, ja que desconcerten a l'usuari i donarien una mala impressió del nostre treball.

Exemples de projectes:

A continuació s'exposa alguns exemples de projecte. No és obligatori que el triem d'entre aquesta llista, tot i que també són opcions vàlides. La seva descripció ens servirà per donar-nos una idea de com pot ser el nostre projecte.

- **Calendari de curs:** Aplicació web que mostra un calendari, amb opció de selecció anual, mensual, setmanal i diària, i on cadascun dels dies indiquen els principals events que habitualment són programats des del principi de curs (reunions periòdiques de tutors o equips docents, dates d'avaluació, períodes de vacances i festius, etc.). Les dades dels events s'han de trobar en un únic fitxer, de tal manera que faciliti qualsevol canvi amb agilitat i facilitat.

- **Notes alumnes:** Aplicació web que permeti mostrar les notes dels alumnes que hi ha desades en un únic fitxer de dades. Introduint una dada identificadora de l'alumne ha de mostrar la llista de qualificacions parcials obtinguda en una determinada assignatura.

- **Qüestionaris:** Aplicació web que mostra un conjunt de preguntes tipus test, generades aleatòriament d'entre un conjunt ampli que es troba desat dins un únic fitxer i convida a seleccionar les respostes correctes. El resultat s'ha d'enviar al correu electrònic del professor i/o s'ha de mostrar directament sobre la pantalla de l'usuari indicant la valoració obtinguda.

- **Àlbum d'imatges:** Aplicació web per mostrar múltiples imatges, classificades i comentades, corresponents als events habituals del centre, com viatges fi de curs, festes escolars, activitats, etc. La relació d'imatges i els estils que s'han d'aplicar s'ha de trobar referenciada en un fitxer, que serà l'únic a modificar si s'ha de crear un nou àlbum.

- **Automatitzar la revista escolar:** Es tracta de crear l'estructura (arxius .js i .css) que ens permetin editar diversos números d'una revista escolar o web del centre canviant, únicament, el contingut de les pàgines. Això significa que, tant els estils, com la disposició dels elements, els menús i el sistema de navegació han de ser reutilitzables sense fer-hi, pràcticament, cap modificació. Ho podem acompanyar de diversos formularis, amb validació, que permetin els usuaris posar-se en contacte amb els editors de la web per demanar informació o expressar opinions.

Durant la fase de creació del projecte, aquest ha d'estar penjat dins una carpeta de nom "projecte" situada dins l'espai web facilitat per la SGTI per a la realització del curs. En aquest espai, el tutor podrà fer el seguiment del projecte i veure'l per poder atendre les consultes. La ubicació final del projecte acabat també es farà en la mateixa carpeta.

La carpeta "projecte" s'ha d'haver creat seguint les directrius indicades en les pràctiques 1, 2 i 3 del **mòdul 1** i la seva pàgina principal d'accés s'ha d'anomenar "index.htm". A la pàgina principal del directori arrel del nostre espai web també hi ha d'haver un enllaç que ens porti fins al projecte (conjuntament amb els enllaços als exercicis dels mòduls), tal com està explicat a l'exercici del mòdul 1.

Recomanacions

Per la realització del projecte és important acompanyar-nos d'una bona eina d'edició web. Dreamweaver ens pot ser útil, principalment per fer les maquetacions de les pàgines i per editar text amb gran extensió.

En canvi, per crear i provar el codi JavaScript són més útils els editors de text convencional o el Dreamweaver amb l'opció de "Mostrar vista código". El codi JavaScript s'ha de provar en petits blocs i en fitxers apart. Ens podem ajudar amb les **tècniques de depuració** explicades a la pràctica 3 del mòdul 8. Quan ja hem comprovat el correcte funcionament, l'incorporem a la pàgina web corresponent.

Per facilitar la reutilització de les pàgines hem de fer que el seu contingut sigui independent de la pàgina contenidora. Podem crear un fitxer de nom *dades.js* que contingui un array i/o variables globals amb dades com imatges, textos, events, etc. i després els enllacem a la pàgina contenidora tal com s'explica als apartats 1 i 2 de la **pràctica 4** del mòdul 8. Amb els estils també farem una cosa semblant.

Si hem separat adequadament les dades de les pàgines contenidores, aconseguirem la màxima reutilització, ja que en properes modificacions solament farà falta canviar les dades i els estils que estaran en fitxers separats.

Proposta

A l'hora de presentar la proposta, hem de fer una pàgina web amb aquestes dades:

Identificació. Nom i cognoms, adreça de correu, títol de la web i tema.

Descripció. Resum dels continguts, breu.

Esquema. Seccions, pàgines i relació entre elles.

Components. Aquesta és la part més important. Hem d'explicar quins tipus de components i recursos de cada tecnologia pensem utilitzar, per què, i com.

Interfície. Més que l'aspecte gràfic, interessa explicar la lògica i la funcionalitat, com es lliguen els diversos components i quines estratègies utilitzem.

Calendari del projecte

E v e n t	D a t a
Proposta	dimarts, dia 28 de gener de 2003
Exercicis del **mòdul 7** (al projecte)	dimarts, dia 11 de febrer de 2003
Exercicis del **mòdul 8** (al projecte)	dimarts, dia 25 de febrer de 2003
Revisió i seguiment	dimarts, dia 25 de març de 2003
Lliurament i fi	dimarts, dia 8 d'abril de 2003

Guia del curs

L'objectiu és conèixer els llenguatges de la web - HTML, css i JavaScript -, que ens han de permetre fer o adaptar pàgines per tal que siguin més interactives i dinàmiques.

El curs s'organitza en 8 mòduls, de 6 hores cadascun, més 12 hores addicionals per acabar el projecte, en total, 60 hores. Cada mòdul consta d'una presentació, quatre pràctiques i exercicis. A cada pràctica hi ha, a més dels conceptes, diversos exemples amb solucions.

1.- Coneixements previs

Per seguir aquest curs s'han de tenir unes destreses avançades en l'entorn Windows, sobretot, amb l'explorador: crear, esborrar i moure carpetes i fitxers; retallar i enganxar, instal·lar i configurar programes, ... Convé haver fet el curs sobre Windows i, encara millor, el de xarxes. És interessant, tenir pàgina web pròpia i/o mantenir la pàgina del centre.

També hem de dominar l'editor de pàgines Dreamweaver i tenir coneixements suficients sobre HTML. Si no els tenim, abans de continuar hauríem de fer una bona repassada dels cursos D72 i D98 sobre Disseny i Creació de Pàgines Web.

Si es coneixen altres llenguatges de programació, millor. Naturalment, també hem de saber navegar per Internet, enviar correu electrònic i trametre fitxers al servidor. El curs ens resultarà més fàcil si tenim, a més, algunes habilitats. La capacitat d'estructurar i organitzar dades i materials resulta imprescindible. També, una bona lògica i un xic de meticulositat.

2.- Requeriments tècnics

L'ordinador ha de tenir instal·lat i ser capaç d'executar l'editor Dreamweaver i, com a mínim, un parell dels navegadors més populars: Microsoft Internet Explorer, Netscape Navigator o Mozilla. Tots han de poder executar JavaScript i tenir les cookies activades.

Tot i que DHTML no està lligat a cap plataforma, el curs planteja els exemples basant-se en el sistema operatiu Windows, que hi ha a la majoria de centres, i dóna per suposat que tenim connexió a Internet i un espai al servidor de la XTEC.

3.- Interfície del curs

Sistema de navegació:

Doble, a l'inici i a la fi de cada pàgina. Quan el cursor se situa sobre una opció de menú, es mostra un missatge amb el títol de la pàgina. A la zona superior, el logotip del curs és un vincle amb la portada. Al peu de la pàgina hi ha un botó per tornar al principi.

Totes les opcions es mostren sobre la mateixa finestra del navegador, excepte la pàgina d'eines, que s'obre en una finestra nova, i l'índex-cercador, que crea una finestra al marge dret de la pantalla. Els resultats dels exemples s'activen amb un botó i es mostren en una finestra tipus "pop-up", superposada a la pàgina.

Índex i cercador:

El curs disposa d'un índex amb cercador intern que permet localitzar ràpidament qualsevol concepte. S'activa des del botó **Índex** del menú de cada pàgina. Si no hi ha cap condició de cerca establerta, es mostra tot el temari. Quan es fa clic sobre un vincle, la pàgina destí substitueix la que teníem a la finestra principal del curs.

Per localitzar una paraula, l'escrivim al quadre de text i fem clic en el botó "Cerca" o premem Intro. El sistema només localitza paraules: si n'escrivim més d'una, mostrarà les pàgines que en continguin alguna. Si no troba res, es fa una segona cerca, més imprecisa.

Eines:

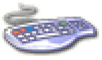
Hi ha algunes eines, fetes específicament per a aquest curs, que podem carregar fent clic sobre el botó **Eines** de la barra de menús. El seu objectiu és doble: per una banda, tenen una utilitat pràctica que veurem al llarg del curs. Per l'altra, s'han dissenyat utilitzant només DHTML i JavaScript, raó per la qual poden ser objecte d'estudi.

Versió per imprimir:

La interfície detecta automàticament quan s'envia algun document a la impressora, de manera que adapta les fonts i elimina els elements de navegació. Cada pàgina té un botó d'impressió.

Significat de les icones:

Informació.
Documentació sobre el contingut que s'està tractant a les pàgines.



Mans al teclat.
Treball que l'alumnat ha de realitzar seguint les propostes guiades.

Qüestions.
Problemes plantejats i resolts a les pràctiques.

Exercicis.
Problemes plantejats que ha de resoldre l'alumnat.

Atenció.
Trucs i suggeriments complementaris a la documentació i a les pràctiques.

Dreamweaver.
Informació o pràctiques sobre l'editor.

Imprimir.
Enviar a la impressora la versió imprimible de la pàgina actual.

4.- Sintaxi dels llenguatges

Tot i que utilitzarem l'editor Dreamweaver, aquest curs ensenya a programar. Per començar, el nostre nivell en HTML ha de ser bo, és a dir, que coneixem la majoria de les etiquetes, sabem el que fan i com s'utilitzen.

Es dóna per suposat que hem utilitzat estils amb el Dreamweaver i també algunes de les funcions JavaScript que incorpora, de manera que estem familiaritzats amb els signes que utilitzen. A continuació en fem un resum.

4.1.- HTML

És un llenguatge de marques, basat en etiquetes, que crea contenidors per situar-hi bocins d'informació. La mateixa paraula serveix per indicar que comença un contenidor i, amb una barra al davant, que s'acaba: <body> … contingut del cos … </body>.

Un contenidor pot tenir, al seu torn, altres contenidors i, aquests, uns altres. L'única condició és que l'últim en obrir-se sigui el primer en tancar-se. Les etiquetes poden tenir modificadors que especifiquen algunes característiques.

HTML és insensible a l'ús de majúscules i minúscules. Dreamweaver codifica sempre en minúscules, que són més fàcils de llegir. Aquest curs segueix el mateix criteri.

4.2.- CSS

Els fulls d'estils css són un afegit que permet manipular els elements d'una pàgina de forma més acurada que HTML. Es basen en la definició de característiques d'aquests elements. Podem redefinir contenidors propis de HTML (body, a, td, img, …) o crear estil nous. S'utilitzen sempre lletres minúscules, excepte en els codis de colors.

Cada estil comença i acaba amb unes claus { }. A dins, les diverses propietats s'escriuen separades per punt i coma (;) amb el format: estil { prop1 : valor1 ; prop2 : valor2 ; … }

Els estils personalitzats comencen per un punt (.) quan són genèrics (s'apliquen als contenidors d'una classe), o per un coixinet (#) si es refereixen a una etiqueta individual.

4.3.- JavaScript

JavaScript és sensible a l'ús de majúscules i minúscules, de manera que Nom i nom són dues variables diferents. Com a llenguatge de programació té moltes més funcionalitats que els altres dos i, en conseqüència, una sintaxi més complexa.

On posem els scripts?

1.- En línia: <a href="javascript: alert('Hola!')"> ... Si les instruccions són prou curtes es poden associar a elements HTML, escrivint la paraula "javascript" seguida dels dos punts.

2.- A la pròpia pàgina, entre les etiquetes <script language="JavaScript"> i </script>. Aquí, com en el cas següent, al lloc concret on s'hagin d'executar o, si no, al <head>.

3.- En un arxiu extern: en aquest cas, l'arxiu no ha d'incloure les etiquetes anteriors. Des de la pàgina s'enllaça escrivint: <script language="JavaScript" src="arxiu.js"></script>.

Comentaris:

Un comentari és un text que escrivim per clarificar el codi i que el navegador no executarà. Així quan el tornem a mirar, l'entendrem millor. Els podem fer de dues maneres:

1.- // (doble barra) comentari que ocupa només una línia, o un bocí.

2.- ./* aquesta combinació indica que comença un comentari, que pot ocupar més d'una línia i, en conseqüència, hem d'indicat quan s'acaba */

Delimitadors:

1.- Delimitació de sentències: Per separar unes instruccions de les altres, tots els navegadors entenen el salt de línia (Intro) com a delimitador. Ara bé, el signe punt i coma (;) és el delimitador predeterminat en molts altres llenguatges, i JavaScript l'ha adoptat. Podem usar qualsevol dels dos per separar instruccions, o tots dos alhora.

2.- Delimitació de blocs: S'utilitzen les claus { } per indicar els límits un bloc d'instruccions, que es tracta com una unitat. Una funció comença i acaba amb aquestes claus. També un bucle o un condicional. Si un bloc de codi té només una instrucció no cal posar les claus:
if (i == 0) document.writeln('No hi ha res per mostrar'). Els blocs es poden enniuar, de manera que dins d'un bloc n'hi pot haver un altre.

3.- Delimitació de literals: Es tracta de valors constants o variables. No cal delimitar els números i valors lògics: 125, 3.1416, false. En canvi, sí que s'han de delimitar les cadenes de text: 'Hola, dia', "Adéu, nit". Podem usar cometes simples (apòstrof) o dobles, tot i que és un bon costum usar les simples per al JavaScript i les dobles per al codi HTML: document.write('Carregar un arxiu');

> Si, en algun moment, la cadena que ja va entre cometes n'ha d'incloure unes altres o un apòstrof, podem usar l'antibarra \ per indicar que el que escrivim a continuació forma part del literal i no pas de la instrucció: document.write('Cadena que inclou l\'apòstrof');

4.- Ús de parèntesis ( ): Tenen dues utilitats: Primer, com en el llenguatge habitual i matemàtic, indicar en quin ordre s'han d'executar les operacions: ((6 + 4) / 2) - 3) dóna 2. També serveixen per indicar els paràmetres d'una funció: alert('Hola món').

5.- Llistes d'elements: Podem utilitzar la coma (,) per separar llistes d'elements, per exemple, en la declaració de variables: var n = 0, i = 0, x = 99;. També per separar elements d'un vector (Array): var aLaboral = new Array('dilluns', 'dimarts', 'dimecres', 'dijous', 'divendres');

6.- Índex d'un element: Els claudàtors [ ] s'utilitzen per identificar un element d'una col·lecció, objecte o Array. El primer element és el zero [0]. En el cas anterior, aLaboral[3] és 'dijous' i document.images[0], la primera imatge d'una pàgina. L'índex no ha de ser forçosament un número. Com veurem al llarg del curs, els identificadors poden fer la

mateixa feina:
document.images['primera'] també val, si havíem escrit <img name="primera" ... >.

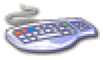
7.- Delimitació d'objectes: El punt (**.**) s'utilitza per indicar jerarquia, és a dir, que un objecte, propietat o mètode forma part d'un altre que té un status superior o fa de contenidor (llegit de dreta a esquerra): document.forms[0].nom.value indica el valor del camp nom del primer formulari contingut al document.

5.- Tutoria

Cada participant tindrà un tutor o tutora que l'orientarà en el seu treball. Podem formular les consultes i preguntes que calgui per correu electrònic. D'aquesta manera també podrà demanar orientacions, suggeriments o recomanacions per desenvolupar el projecte.

La persona responsable de la tutoria farà un seguiment personal del procés de cada participant en base als exercicis. Les pràctiques es poden trametre a la carpeta del servidor, tot i que no és imprescindible, excepte quan s'especifica expressament.

6.- Projecte



A partir del mòdul 7 haurem de començar a treballar en un projecte final, l'elaboració d'una web, on posarem en pràctica els coneixements del curs. Aquest és l'element avaluador definitiu per obtenir el certificat.

La web del projecte ha de ser d'interès educatiu, lliure de motivacions comercials. El contingut es regeix per la mateixa **normativa** que les pàgines personals i les webs de centres hostatjades en el servidor de la XTEC. El contingut de les pàgines ha de ser original. Si s'utilitzen imatges o textos aliens, cal citar la font. Les imatges o textos citats no poden superar mai el 10% de l'obra. El projecte és un treball personal i no es pot fer en equip.

El calendari estableix una data per al lliurament del projecte i una altra per al lliurament d'una versió revisada. Cal entendre que el projecte ha d'estar enllestit a la primera data. Només si el tutor considera que s'han d'efectuar modificacions perquè no es compleixen els requisits, el podrem corregir i lliurar-lo la data termini de la versió revisada.

7.- Calendari

Vegeu calendari de l'activitat.

Eines i recursos

Aquestes eines ens poden ajudar en la realització del curs.

Eines

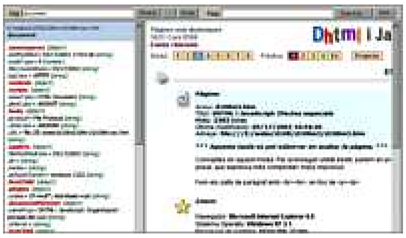
Provador de codi JavaScript. [obrir]

Permet escriure i provar el codi dels scripts. També resulta útil per veure com funcionen els scripts agafats d'Internet, abans de guardar-ne una còpia. Cal marcar el codi a la pàgina origen i prémer *Control+C* (Copiar). Després, ens situem al quadre d'addició, premem *Control+V* (Pegar) i el provem. Si el codi conté un "onload()", s'haurà d'actualitzar la pàgina.

FerÍndex: generador d'índex [obrir]

Si tenim una carpeta amb pàgines numerades (p.e. pag01.htm, pag02.htm, ...) aquesta eina genera una pàgina amb l'índex. Per això, va obrint els arxius i en memoritza el nom i el títol. El codi que genera (HTML i JavaScript) es pot editar després amb Dreamweaver. Incorpora un provador de codi amb full d'estils.

Inspector d'objectes JavaScript. [obrir]



El navegador crea, per a cada pàgina, una estructura jeràrquica d'objectes (dades i accions), anomenada Document Object Model (DOM). JavaScript pot manejar el DOM. L'inspector mostra el seu contingut i organització. Molt útil per veure les diferències entre navegadors i les possibilitats de cadascun.

Recursos

Referència JavaScript 1.3 de Netscape:
[http://developer.netscape.com/docs/manuals/js/client/jsref/index.htm]

La web dels creadors de l'invent ofereix una referència completa sobre cadascun dels elements del llenguatge. En anglès, és clar. Ep! l'adreça és d'Internet, i pot canviar.

Cerca a Google: [google.com]
Al món hi ha milers de pàgines!

dhtml javascript css >>